UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

Integración continua para open hardware

Proyecto fin de carrera en Telefónica I+D

Universidad
Carlos III de Madrid

David Del Peral Chico

Madrid, Diciembre del 2012

Tutor UC3M: **David Expósito Singh**
Tutor Telefónica: **Francisco Javier Zorzano Mier**

Agradecimientos

Este proyecto representa el final de una etapa, en varios ámbitos, y el comienzo de una nueva. Por este motivo, quisiera agradecer a las siguientes personas, el apoyo proporcionado y la confianza depositada a lo largo de mi carrera universitaria.

En primer lugar, se encuentra mi familia, mis padres José Luis y María de los Ángeles y mi tía Regina, así como el resto de familiares, que me han brindado consejo en momentos de duda y han tenido la gentileza de ayudarme en cualquier tipo de dificultad que se presentase. Por supuesto, también tengo palabras para aquellos que nos han dejado, y que siempre intentaron aportarme su sabiduría cultivada: mi abuela Ángeles y mi abuelo Estanislao. Un gran apoyo moral, que siempre ha conseguido sacarme una sonrisa en momentos difíciles, es Marta, a la cual, también agradezco su intención de compartir conmigo su visión.

Una parte importante de cualquier persona, siempre es influenciada por sus amigos y conocidos, compañeros en la vida, con los cuales se comparten los buenos y los malos momentos. Por esta razón, este proyecto va dedicado, de forma muy especial, a todas esas personas que han sido protagonistas en mi vida, mis amigos de Villalba Del Rey (Cuenca) y alrededores, tales como Kevin, Vidal, Fran, Raúl Salgado, Estefanía, Melody, Judith, Raúl Gascueña, David Romero, Víctor Huete, Penélope, Alberto Alcocer, Armando, Diego Batanero, etc., sois tantos que es imposible nombraros a todos, pero tanto los que nombro como los que faltáis sois igual de importantes.

En el lugar donde transcurre mi vida diaria, quisiera realizar una especial mención a Víctor Nieto, por su apoyo y ayuda, y por ser un compañero y amigo fiel a lo largo de más de 15 años de amistad. Por supuesto, también tengo palabras para su novia Bea, por querer siempre aconsejarme en aspectos en los cuales no soy muy hábil. No me olvido de Honorio García, Javier Fombellida, Javier Rojas, Eduardo Muñoz, Roberto Estrada, Javier Castrejón, Alberto, Sohrab, Cristian Galán, Pedro Palencia, Íñigo Mariño, Cristina Madrid, etc., grandes personas que han sabido aguantarme.

Este proyecto ha sido posible gracias a la confianza que Telefónica y Francisco Javier Zorzano Mier han depositado en mí. Una gran oportunidad para allanar el camino de mi futuro. Por otra parte, gracias a mis compañeros y excompañeros de trabajo por aportarme su experiencia, ayudarme en mi labor, y soportarme en el día a día como pueden: Carlos Merchán, Pablo Picazo, Alejandro Veleiro y Javier Carazo.

Por último, y no menos importante, doy las gracias a mi universidad, su profesorado y personal de administración, por la vida universitaria que tanto añoraré. Gracias a mi tutor, David Expósito Singh, por sus consejos, apoyo y gran involucración en este proyecto.

En definitiva, este proyecto es para todos los que han creído en mí. Gracias.

"*Your most unhappy customers are your greatest source of learning*."
Bill Gates. 1999.

Resumen

En estos últimos años, la informática, y más concretamente, el hardware, está evolucionando hacia sistemas empotrados. La aparición de nuevos mercados como los microrordenadores, televisiones inteligentes, etc., y masificación de los existentes como los teléfonos inteligentes y las *tablets* amplifica este fenómeno. Esto es debido a las ventajas de dichos sistemas en cuanto a coste a escala, optimización y rendimiento, consumo de energía o tamaño, entre otras.

Los sistemas empotrados crecen en soluciones cerradas de forma exponencial, pero lo más interesante, es la mezcla resultante entre la filosofía libre y los sistemas empotrados. De esta manera, están surgiendo comunidades y empresas de *open hardware* que están logrando, cada vez más, atraer la atención de los medios de comunicación, universidades y empresas. El motivo es bien sencillo: estas comunidades o empresas aportan libremente plataformas de computación y control de coste bajo y facilidad de programación, además de extremadamente flexibles. Este fenómeno ocasiona que surjan iniciativas de innovación basadas en estos tipos de plataformas, añadiendo una conexión a internet, lo cual aumenta exponencialmente las posibilidades de uso y explotación, naciendo así conceptos como el internet de las cosas. Además, la variedad que está logrando el *open hardware* no tiene antecedentes: desde microrordenadores como Raspberry Pi hasta plataformas de control como Arduino, en gran variedad de arquitecturas (x86, ARM, AVR, PIC, etc.).

Por otra parte, cada vez es más patente el desarrollo ágil en las empresas e instituciones por la flexibilidad que permite, entre otras ventajas. El desarrollo ágil se encuentra en constante evolución y comienza a expandirse a otras áreas fuera de la implementación, como son las pruebas de calidad. A partir de aquí, nace el concepto de integración continua a través de pruebas de regresión, una nueva forma de automatizar pruebas unitarias de forma ágil que asegure la correcta funcionalidad del código fuente desde la implementación de la primera funcionalidad.

El concepto de la integración continua es aplicado al software en general, pero, nadie se había propuesto aplicarlo en el software empotrado, a través de pruebas de regresión que comprueben la correcta ejecución del programa sobre el hardware real, posibilitando la monitorización directa sin simulaciones. De esta manera, se puede establecer un entorno de trabajo ágil en la programación de software o firmware para microcontroladores o microprocesadores sencillos, así como su futura aplicación a otras plataformas.

En este proyecto, se tratará de proporcionar una solución real de integración continua para una plataforma de *open hardware*: Arduino. Esta solución se deberá integrar dentro de un sistema de integración continua tradicional, lo que posibilitará la utilización de las funciones de planificación, actualizaciones desde un sistema de control de versiones, entre otras, así como su disposición junto a otros proyectos de integración continua de software. Esto se realizará mediante un programa de adaptación, compatible con dichos sistemas, que se encargue de proporcionar el soporte al hardware (carga, compilación, etc.) y monitorice su comportamiento.

Abstract

In recent years, information technology and, more specifically, hardware, are shifting to embedded systems. This phenomenon is increasing because of new emerging markets such as microcomputers, smart TVs, smartphones and tablets, among others. Some of the main reasons of this change are the benefits of the embedded systems in terms of scalability, optimization, performance, power consumption and size.

Currently, the market of embedded systems grows exponentially, and the most interesting synergies appear when merging the concepts of open knowledge and embedded platforms. Nowadays, communities and open hardware companies are emerging and they receive attention from the media, universities and companies. The reason of this is simple: these communities and companies provide free computing and controlling platforms with low cost and ease of programming, as well as extreme flexibility in their market. This phenomenon has motivated the creation of innovative initiatives based on these platforms. One example consists in developing embedded systems with internet connection. This increases exponentially the possibilities of use. In addition, the open hardware has an unprecedented variety of architectures: covering from microcomputers such as Raspberry Pi to control platforms like Arduino, for different architectures (x86, ARM, AVR, PIC, etc.).

Moreover, agile development is increasingly important in companies and institutions. The reason is the flexibility that it allows. Nowadays, agile development is constantly evolving and begins to expand to other areas such as quality testing. By this context, the concept of continuous integration through regression testing appears. This is a new way of automating unit testing which ensures the correct functionality of the source code from a given platform.

The concept of continuous integration is currently applied to generic software, but to the best of our knowledge, no one had proposed to use it with embedded software. The idea is to combine it with regression testing, in order to check the correct execution of the program in real hardware. With this idea it is possible to directly monitor a platform without performing simulations and to create a flexible design environment for software or firmware programming of microcontrollers or microprocessors, including their future extension to other platforms.

In this project, we develop a real continuous integration solution for Arduino, an open hardware platform. This solution is integrated into a traditional continuous integration system, which allows the use of planning features and updates from a source code control system, including features of these systems, as well as their combination with other continuous integration projects. This solution is developed through software compatible with these systems, which is responsible for providing support to the hardware and performing the monitoring of their behavior.

Índice

Índice de tablas

Índice de ilustraciones

1. Introducción

1.1 Explicación de la problemática

El grupo Telefónica es una multinacional de gran influencia y peso en el negocio de las telecomunicaciones, donde la máxima siempre ha sido la comunicación y transmisión de datos. La comunicación fue, en un principio, de personas, pero la tecnología y su avance exigen una nueva comunicación: la comunicación entre objetos, o en su denominación más conocida, el internet de las cosas.

Telefónica cuenta un departamento de innovación y desarrollo dentro de su negocio global Telefónica Digital, en el cual, existe un área de tecnologías emergentes donde se buscan nuevos caminos hacia nuevas formas de usar la tecnología. Uno de los negocios que cada vez cobra más importancia es el mundo M2M, es decir, la comunicación entre máquinas. Actualmente, el negocio M2M sólo se extiende y es rentable para empresas a partir de un determinado tamaño mediano, con costosas soluciones a medida. Por esta causa, uno de los objetivos a cumplir en el futuro es la democratización del internet de las cosas, en otras palabras, que cualquiera, ya sean particulares, autónomos o pequeñas empresas, puedan conectar cualquier objeto a internet e interactúe[1].

Aquí es donde entra en juego Arduino, una plataforma de hardware de filosofía abierta, que cualquiera puede conocer y construir. Uno de los objetivos de Arduino es promover el conocimiento de electrónica a nivel sencillo y que cualquier persona con unos muy mínimos conocimientos sea capaz de montarse un aparato útil y programarlo, trasteando de forma fácil e intuitiva con sus elementos y programando su microcontrolador[2]. Para ello, Arduino facilita tutoriales, esquemas, un lenguaje y librerías sencillas y un IDE pensado para personas con apenas conocimientos de electrónica o informática. Otro punto de Arduino es su poco coste y el gran soporte que ha obtenido, por filosofía y por su comunidad de desarrolladores y usuarios, además de empresas implicadas en el proyecto ofreciendo kits de trabajo, sensores, accesorios, componentes, etc.

Arduino es muy propicio, por los motivos expuestos, para desarrollar nuevos usos e ideas de una manera fácil y barata, y es por ello que, Telefónica posee interés en él, aparte de la imagen y publicidad que dan el involucrarse en un proyecto de código y hardware abierto. Como Telefónica es una compañía global de telecomunicaciones, la mejor forma de colaborar con el mundo de Arduino es proporcionar una solución de conectividad, y así promocionar y ofrecer un internet de las cosas real, con dispositivos Arduino conectados a internet o entre sí, que cualquiera puede utilizar. Por estas razones, Telefónica ofrecerá un complemento o *shield* para Arduino, licenciado por éste como oficial, que le dotará de conectividad GSM/GPRS, así como una SIM y diferentes planes de datos especializados[3].

En este punto, se puede ver el interés de este proyecto fin de carrera. En los equipos desarrolladores, es muy frecuente la utilización de sistemas automatizados de pruebas, que testean la última versión desarrollada compilándola y controlando sus posibles resultados. Estos sistemas poseen varias denominaciones (sistemas de regresión, sistemas de integración continua, sistemas de pruebas unitarias, etc.) y características dependiendo de la plataforma para la cual estén diseñados. Esto proporciona una forma eficaz de controlar los cambios en el software en desarrollo, detectando los errores de programación o resultados inesperados, con lo que, se consigue agilidad y eficiencia en la corrección de dichos defectos. Esta forma de desarrollo es conocida como *Test-Driven Development*.

Test-Driven Development es una técnica ágil para desarrollar software de forma incremental. Cada parte de código que se finaliza siempre debe ser testeada. Los test tienen la característica de ser pequeños, es decir, pruebas partes simples y de una funcionalidad muy básica, cerciorándonos así de qué partes se encuentran desarrolladas correctamente. En caso de que un test falle, rápidamente se detecta el lugar, y, al ser pruebas unitarias, la corrección suele ser sencilla y eficaz. El software es algo muy frágil, pequeños cambios pueden provocar grandes consecuencias y afectar a varios módulos, clases, componentes, etc., es por ello, que las pruebas deben ser sencillas y de tamaño moderado para que abarquen todas las posibilidades de ejecución distinta del código. En las tradicionales baterías de test estructuradas no se tiene la certeza de haber contemplado todas las posibles combinaciones de uso del software[4][5].

Existen paquetes de software y entornos orientados a esta técnica: software de automatización como Hudson/Jenkins o CruiseControl, complementos o módulos de pruebas para diferentes lenguajes de programación como JUnit para Java o Unittest para Python, soluciones de gestión de proyectos como Apache Maven, entornos de desarrollo como Eclipse, Visual Studio o NetBeans y sus correspondientes *plugins* de integración, etc. La combinación de *plugins* para IDEs, módulos de testeo y sistemas de automatización posibilita la construcción de un sistema robusto que cumpla con las directrices de Test-Driven Development. Además, el uso de un repositorio de versiones como SVN, GitHub o Mercurial acentúa esta manera de desarrollo, aparte de cumplir con la mayoría de las normativas de desarrollo impuestas por las empresas que utilicen metodologías ágiles, como Telefónica en este caso.

El contrapunto con los intereses de Arduino y Telefónica es que esta técnica sólo está aplicada al software, es decir, no existe una solución pensada para testear hardware y su firmware/software empotrado. Existen algunos autores que han escrito textos y libros proporcionando una visión teórica con ejemplos sobre técnicas en el software empotrado[5], pero en el mercado, no existe una solución completa, y menos aún, especializada en Arduino.

El creciente interés del internet de las cosas en la sociedad, la oportunidad de empresas como Telefónica para realizar negocio y expandir las fronteras de la comunicación, la popularización de Arduino como plataforma abierta hardware pensada para el usuario básico y limitado en conocimientos y la gran comunidad que sostiene estos pilares son los que justifican la existencia de un sistema de pruebas automatizado para Arduino utilizable por empresas y desarrolladores que ahorre tiempo y mejore la eficiencia del desarrollo de firmware o software empotrado, controlando en todo el momento el comportamiento del microcontrolador.

Tanto es así, que este proyecto fin de carrera posibilita la existencia de un desarrollo ágil de software y firmware para Arduino, testeando el comportamiento directamente en la placa hardware, sin simulaciones. Es por ello, que es utilizado por Telefónica para realizar las pruebas de sus librerías y proyectos para Arduino, siendo compatible con su política de versiones.

1.2 Objetivos del PFC y estructura del documento

Una vez expuesta la problemática y el entorno actual de la empresa, se procede a desarrollar los objetivos del proyecto fin de carrera.

1.2.1 Objetivos principales

Desde el punto de vista de la empresa, esta puede disponer de solución integrada de automatización de pruebas para software empotrado en un microcontrolador, en este caso, siguiendo la arquitectura abierta de Arduino, controlando sus resultados y monitorizando los pines de comunicación. Gracias a esta solución, se posibilita lograr otro objetivo: el desarrollo de software en hardware empotrado a través de metodologías ágiles de una forma eficaz.

Desde el punto de vista de la comunidad y la investigación, este sistema cubrirá un espacio hasta ahora poco desarrollado: la automatización en hardware y su monitorización, aportando una solución real a la comunidad Arduino, y que puede ser un punto de partida para futuros sistemas adaptados a otro tipo de hardware con software empotrado.

1.2.2 Objetivos específicos

A parte de los objetivos principales, también se establecen unos objetivos específicos, imprescindibles para que los objetivos principales sean posibles. Estos objetivos se tratarán a lo largo del desarrollo del proyecto:

- **Desarrollo de una herramienta de gestión ágil para la compilación y carga del software en las placas hardware de Arduino:** es un aspecto fundamental del proyecto, esta herramienta es el núcleo de nuestra gestión y es lo que se ha denominado Testduino. Esta herramienta deberá gestionar el proceso de test y todo lo que ello implica: rutas de acceso y configuración de parámetros de transmisión y compilación, comunicación serial, coordinación de placas Arduino para testeos duales, acceso y ejecución del software de compilación y carga para el microcontrolador, localización de fallos y errores y notificación de los mismos. La herramienta debe presentar las siguientes características:
    - **Escalabilidad:** es imprescindible que la herramienta sea escalable, es decir, se pueda utilizar con un número de placas Arduino considerable.
    - **Optimización:** una herramienta de estas características se suele ejecutar automáticamente en segundo plano junto a otros muchos procesos, por lo que, el consumo de recursos del sistema operativo y el hardware no debe ser excesivo.
    - **Fiabilidad:** los resultados deben de ser fiables en una herramienta de detección de errores como ésta, además de ofrecer información detallada de su ejecución.
    - **Reusabilidad:** dados los intereses empresariales y de la comunidad, es casi una obligación, y con mayor motivo en una plataforma de filosofía abierta, ofrecer una programación sencilla de comprender, fácil de modificar y reutilizable. El software debe estar estructurado con vistas a facilitar su ampliación y mejora en un futuro.

  - **Integración:** es una característica fundamental la correcta integración de la herramienta en entornos de planificación de tareas o de pruebas automáticas: gestores de tareas, integración continua, programadores, etc.

- **Desarrollo de una librería que posibilite la monitorización dual:** con esto nos referimos a que un Arduino monitorice las salidas digitales de otro, para que, más adelante, la herramienta de gestión (Testduino) compruebe que los valores son los adecuados. Esta librería debe ser compatible con el API de C/C++ de Arduino para su correcta ejecución, aparte de soportar diferentes tipos de placas con diferentes números de pines.

- **Integración de una solución única funcional:** utilizando el sistema de integración continua Jenkins (o su predecesor Hudson) como base para la gestión, planificación y registro de historiales, lograr una integración de la herramienta Testduino con dicho sistema, muy recomendado para este tipo de tareas. Además, se incluirá dentro de la solución un repositorio de versiones donde alojar las pruebas a realizar y las librerías requeridas y/o desarrolladas para probar también.

1.2.3 Estructura del documento

Con el objetivo de facilitar la lectura del proyecto fin de carrera, se ofrece un breve resumen descriptivo de las partes que componen su documentación:

- **Introducción:** capítulo introductorio donde se aborda la problemática presente, los objetivos del proyecto fin de carrera, así como la estructura de su documentación.

- **Estado de la cuestión:** descripción de la situación actual y de las alternativas en el mercado en cuanto al software y el hardware necesario. También se comentará la situación empresarial de Telefónica respecto al material del proyecto.

- **Análisis de la solución:** estudio de las necesidades planteadas y definición de los requisitos y casos de uso apropiados.

- **Diseño de la solución:** descripción en profundidad de la solución, tanto de las librerías como del programa de adaptación, su interacción con el hardware en los procesos de compilación, carga y monitorización, su composición y clases y su desarrollo y arquitectura. Finalmente, se incluyen las matrices de trazabilidad que relacionan los requisitos entre sí, y éstos con los componentes resultantes.

- **Pruebas y evaluación:** presenta las pruebas que se realizarán a la solución desarrollada para verificar su correcto funcionamiento. Además, se realizará un estudio detallado del impacto de la solución en el rendimiento de Arduino y otros factores involucrados.

- **Planificación:** este capítulo describe la planificación del desarrollo realizada con un software de predicción de costes y tiempo.

- **Presupuesto:** este capítulo está destinado a abordar el cálculo de un presupuesto detallado que conjunte todos los costes necesarios para la realización del proyecto.

- **Conclusiones y trabajos futuros:** resumen de las conclusiones que se obtienen del estudio de la solución y datos observados en los análisis de este documento. Se incluyen comentarios e ideas sobre futuras expansiones o mejoras para la solución, que no se han realizado por exceso de tiempo de desarrollo.

- **Bibliografía:** enumeración de referencias utilizadas a lo largo del documento.

2. Estado de la cuestión

En este capítulo, se aborda el estado actual del mercado, la situación del software externo a utilizar por la solución, el hardware en sí y las diferentes alternativas existentes. También se incluye el uso de estos elementos que se realiza en Telefónica actualmente.

2.1 Software de control de versiones

Un sistema software de control de versiones son un grupo de aplicaciones pensadas y desarrolladas para gestionar de forma ágil los cambios en el código fuente de un desarrollo actual, y poder revertirlos con seguridad. Este ámbito, últimamente, ha sido ampliado desde el concepto control de versiones, llegando al concepto software *configuration management*, en el que se engloban todas las actividades que pueden realizarse por un equipo sobre un desarrollo, englobando documentos, ofertas, dibujos, esquemas, etc.

Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas[7][6].

Los orígenes del sistema provienen de la época en que apenas existían redes de ordenadores. En esta época, los ficheros se gestionaban de manera individual, y ya se encontraban algunas herramientas predecesoras:

- **Revision Control System (RCS):** almacenaba la última versión y sus diferencias con la anterior. Permitía la recuperación eficaz del fichero origen.

- **Source Code Control System (SCCS):** gestiona diferencias entrelazadas entre ficheros, que permiten generar versiones como un conjunto de sub-revisiones. Fue pionero y formó parte por UNIX.

2.1.1 Ventajas e inconvenientes

Disponer de un sistema de control de versiones proporciona múltiples ventajas:

- Tener un **control absoluto** de todas las versiones, de cuál es la versión más reciente, y quién y cuándo la ha actualizado.

- **Comparar versiones,** sabiendo de esta manera cuales son los cambios realizados.

- **Volver atrás en nuestros desarrollos** cuando estos no nos ofrecen los resultados esperados o simplemente cuando encontramos errores.

- Tener **distintas ramas del proyecto**, es decir, ramificar diferentes funcionalidades o versiones de un software y desarrollarlas por separado.

- **Ahorro de espacio** en disco.

Las únicas **desventajas** a mencionar dependen del modelo de funcionamiento que escojamos, así como del **coste y riesgos del soporte** asociado a mantener los repositorios.

2.1.2 Modelos de funcionamiento

Actualmente, y gracias a la aparición de las redes de trabajo existen dos modelos de funcionamiento:

- **Modelo cliente-servidor:** los desarrolladores utilizan un servidor común, conocido como repositorio, al que acceden a través de una aplicación cliente.

- **Modelo distribuido:** los desarrolladores poseen un repositorio local, los cambios son compartidos posteriormente.

2.1.2.1 Ventajas del modelo cliente-servidor

- **Fuerte control del desarrollo** al disponer de una **versión centralizada** del proyecto, lo que puede facilitar la coordinación de los desarrolladores.

- Versiones identificadas por un **número de versión**, lo que facilita la unificación de versiones y seguimiento de cambios realizados.

2.1.2.2 Ventajas del modelo distribuido

- **No necesita estar conectado a la red** salvo para actualizar las modificaciones. Esto produce una mayor **autonomía y rapidez**.

- Si se cae el servidor remoto, los **desarrolladores puede seguir editando su copia local**.

- Al disponer de copias locales en cada equipo de cada desarrollador, **la información está replicada y el sistema posee más posibilidades de recuperarse** en caso de desastre o error grave. En otras palabras, no es estrictamente necesario realizar *backups*, aunque siguen siendo necesarios para resolver situaciones en las cuales la información perdida no ha sido replicada aún.

- Permite mantener **repositorios más limpios**, debido a que cada desarrollador decidirá si es relevante compartir ciertos cambios, lo que facilita la gestión de versiones inestables o *tags* propios.

- El servidor remoto necesario requeriría **menos recursos** que un servidor centralizado.

- Son **sistemas más recientes**, por lo que proporcionan mayor **flexibilidad y escalabilidad**, además de su constante evolución tecnológica actual. Un ejemplo es el uso de ramas en repositorios locales para realizar pruebas, versiones inestables, trabajo diario, etc.

2.1.3 Funcionamiento elemental

Los sistemas permiten elegir qué ficheros gestionar, cuándo actualizar los repositorios o deshacer los cambios (estos pueden ser modificaciones parciales, añadir, borrar, renombrar o mover elementos), y extraer cualquier estado anterior gracias al registro histórico de las acciones realizadas por elementos y usuarios. Además, ciertos sistemas incorporan tareas avanzadas como generación de informes de los cambios entre versiones. Normalmente, a la última versión se le suele acuñar el término HEAD.

Existen dos formas de edición posibles:

- **Exclusiva:** para editar un fichero, el desarrollador comunica al repositorio el elemento que se desea modificar y el sistema se encargará de impedir que otro desarrollador pueda modificarlo durante ese instante. Cuando se ha terminado la edición, el fichero se libera y se comparte.

- **Colaborativa:** al editar un fichero, el desarrollador modifica su copia local, la cual deberá compartir al finalizar (realizando un COMMIT). Entonces, el sistema intenta compartir y combinar las diversas combinaciones. Esta forma conlleva un problema de conflictos que deben ser solucionados manualmente en ocasiones, además de, posibles inconsistencias si existe falta de coordinación entre desarrolladores. Esta semántica no es apropiada para ficheros binarios.

2.1.3.1 Flujos de trabajo

Una de las características más interesantes de los sistemas de control versiones es la posibilidad de asentar flujos de trabajo en el grupo de desarrollo. Estos flujos de trabajo indican cómo se relacionan los diferentes usuarios para colaborar entre sí. Los sistemas centralizados (modelo cliente-servidor), como es lógico, restringen de buena manera la forma en la que los usuarios colaboran, en cambio, los sistemas distribuidos permiten una gran flexibilidad incorporando el concepto de que cada desarrollador puede contribuir y recibir contribuciones, lo que evidentemente resulta más colaborativo. El utilizar diversos flujos de trabajo permite trabajar con gestores de integraciones para ciertas partes críticas o de especial relevancia[6].

A continuación, se analizan algunos ejemplos comunes:

- **Flujo de trabajo centralizado:** cada desarrollador consiste en un nodo de trabajo. Por otra parte, existe un repositorio remoto central que funciona a modo de punto de sincronización. Existe total igualdad entre nodos de trabajo desde el punto de vista del repositorio central. Por otra parte, en un sistema distribuido, cada nodo se correspondería con un repositorio local de cada desarrollador. Existe una gran desventaja: si dos usuarios clonan y editan un fichero, sólo el primero que realice estas acciones podrá actualizar el fichero central limpiamente, es decir, el segundo deberá fusionar su trabajo previamente con el del primero antes de enviarlo para evitar sobre-escribir los cambios del primero, esta situación se conoce como *non-fast-forward changes*.

- **Flujo de trabajo con un gestor de integraciones:** cada desarrollador tiene acceso de escritura a un repositorio público propio y acceso de lectura a todos los repositorios públicos de los demás usuarios. Por otra parte, es necesaria la existencia de un repositorio canónico o maestro, que sirve como base. La contribución se basa en que cada desarrollador crea su clon local del repositorio canónico y envía los cambios a él, entonces, a la hora de actualizar los cambios definitivamente al repositorio canónico, cada desarrollador tiene que realizar una petición a la persona gestora del mismo. Este flujo posee una gran ventaja: siempre podrás continuar trabajando sin estar a expensas de cambios de otras personas, es decir, podrás llevar tu ritmo de trabajo, ya que la persona encargada del repositorio canónico podrá recuperar tus cambios siempre.

- **Flujo de trabajo con dictador y tenientes:** en realidad es una ampliación del flujo de trabajo con gestor de integraciones. Es utilizado en proyectos muy grandes, con cientos de colaboradores, lo cual se aleja un poco de la visión de grupo de desarrollo empresarial, pero este método ha demostrado ser efectivo y cuenta con la experiencia de uso en el desarrollo del *kernel* de Linux. Consiste en que existen una serie de gestores de integración que se encargan de partes concretas del repositorio conocidos como tenientes. Todos los tenientes están sumisos al gestor de integración principal, conocido como dictador. El dictador integra todos los aportes de los tenientes publicando el trabajo en un repositorio de referencia del que recuperan todos los desarrolladores. En otras palabras, este flujo permite que el gestor de integración pueda relegar trabajo en sus tenientes.

2.1.3.2 Ramas

Las ramas son una herramienta que nos permite flexibilizar el método de colaboración de los desarrolladores proporcionando diferentes espacios de trabajo con diferentes versiones que agilicen el desarrollo del software. Podemos distinguir entre dos posibilidades básicas: **ramas puntuales o ramas estables de larga duración**.

En las ramas puntuales, éstas representan funcionalidades concretas que necesitan un desarrollo apartado del general. También son conocidas como ramas de soporte, y permiten al desarrollador centrarse exclusivamente en el desarrollo de una característica concreta, y cuando finaliza su desarrollo, se fusiona con el proyecto general o con ramas superiores. La fusión se realiza sólo si se está totalmente seguro de que la característica funciona en cualquier caso, es decir, no se sigue ningún orden. Esto provoca que las ramas superiores no sean "ensuciadas" con modificaciones relativas a una funcionalidad concreta. Existen varios subtipos de ramas puntuales: ramas de nueva funcionalidad (*topic branch* o *feature branch*), ramas para corregir error (*hotfix branch*) o ramas de versión (*release branch*), éstas últimas permiten dar soporte a una nueva versión de producción controlando su contenido y proporcionando mantenimiento, correcciones y mejoras sobre ella.

Por otro lado, las ramas estables o de larga distancia o de largo recorrido suelen tenerse siempre abiertas, representando diversos grados de estabilidad. Normalmente, existe una rama máster totalmente estable (denominada también *stable*) que se encuentra totalmente controlada y disponible para entornos de producción. A partir de este punto, el resto de ramas son más inestables y las podemos calificar como ramas *beta*, *alpha*, *unstable*, *dev*, *experimental*… que son las ramas sobre las que se va trabajando. Cuando se alcanza un cierto grado de estabilidad mediante unos requerimientos establecidos, se fusionan los cambios confirmados con la rama master. Un ejemplo de uso de este tipo de ramas es la forma de organización de las versiones del sistema operativo Debian GNU/Linux[6].

2.1.4 Mercado y alternativas

En la actualidad, existen tanto soluciones de código abierto como de código privado, con diferentes tipos de licencia y filosofías, gratis y de pago, y adaptadas tanto a modelos centralizados como a modelos distribuidos. Veamos las alternativas más utilizadas y conocidas en el siguiente análisis.

2.1.4.1 Concurrent Version System (CVS)

Ilustración 1: Logotipo de CVS

Basado en el modelo cliente-servidor. Se licencia bajo licencia GPL y es gratuito, fue desarrollado por GNU y es muy popular dentro del mundo del software libre. Actualmente se encuentra obsoleto y está siendo sustituido por alternativas más actualizadas como Subversion. Su utilización se extendió debido a la posibilidad de un manejo eficaz mediante comandos desde consola.

Posee algunas limitaciones: los ficheros no pueden ser renombrados y posee un soporte discreto de ficheros Unicode con nombres de archivo no ASCII. Por el contrario, el gran uso que ha tenido y su popularidad hace que exista software que complemente este sistema: versiones para diferentes sistemas operativos, *GUIs*, etc[8].

2.1.4.2 Subversion (SVN)

Ilustración 2: Logotipo de Subversion

Basado en el modelo cliente-servidor. Se licencia bajo licencia tipo Apache/BSD, es gratuito y es también conocido como SVN. Posee una característica importante respecto a CVS: todo el repositorio tiene un único número de versión que identifica un estado común de todos los ficheros en un instante determinado. Es uno de los sistemas más reconocidos por la comunidad del software libre y se utiliza en proyecto de gran envergadura como Apache, Django o FreeBSD. Además, existen servicios de alojamiento que lo proporcionan como SourceForge, Google Code, CodePlex o RedIRIS[9].

Entre sus ventajas figuran: historia de ficheros y directorios a través de copias y renombrados, modificaciones atómicas, creación de ramas y etiquetas con la misma complejidad que en CVS, sólo se envían diferencias (no ficheros completos), puede ser soportado mediante Apache (con todas sus ventajas: LDAP, SQL, etc.) sobre WebDAV, maneja eficientemente ficheros binarios y permite bloqueo selectivo de ficheros. Sus carencias se podrían resumir en que el manejo de renombrado no es completo, es decir, lo maneja como suma de una operación de copia y una de borrado, y no resuelve el problema de aplicar repetidamente parches entre ramas, en otras palabras, no facilita saber con exactitud el orden de los cambios que se han realizado. Como en CVS, su popularidad le ha llevado a contar con multitud de clientes para diferentes sistemas operativos e incluso extensiones para entornos de desarrollo[10].

2.1.4.3 Git

Ilustración 3: Logotipo de Git

Basado en el modelo distribuido. Fue creado por Linus Torvalds, y por lógica, es utilizado en el grupo de programación del núcleo Linux. Actualmente recibe contribuciones de alrededor de 280 programadores. Es gratuito y software libre, utilizando la licencia GPL. Su diseño está inspirado en BitKeeper y en Monotone, ambas alternativas de pago, mezclado con la experiencia de su creador y a la vez creador de Linux, ya que lleva bastantes años manteniendo una enorme cantidad de código distribuida y gestionada por muchos desarrolladores, que incide en numerosos detalles de rendimiento, y de la necesidad de rapidez en una primera implementación.

Sus características se resumen en: fuerte apoyo al desarrollo no-lineal, es decir, rapidez en la gestión de ramas y diferentes versiones, posibilidad de publicación de almacenes de información mediante HTTP, FTP, *rsync* o protocolo nativo ya sea a través de cifrado SSH o sin cifrar, emulación de servidores CVS, uso directo de repositorios Subversion y SVK, gestión eficiente de proyectos grandes por su optimización de velocidad de ejecución en la gestión de diferencias entre archivos, notificaciones obligatoria de cambios posteriores a versiones previas a un cambio, evita coincidencia de ficheros diferentes en un único nombre ya que los renombrados se basan en similitudes entre ficheros, almacenamiento periódico en paquetes y trabajo en base cambios de proyecto en vez de cambios de fichero[11].

2.1.4.4 Microsoft Visual Studio Team Foundation Server

Ilustración 4: Logotipo de Microsoft Visual Studio Team Foundation Server

Basado en el modelo cliente-servidor. Desarrollado por Microsoft. Está integrado dentro del entorno de desarrollo Microsoft Visual Studio, y, por lo tanto, está orientado a desarrollos en tecnologías Microsoft (.NET). Evidentemente, su licencia es EULA y es software privado de pago. Ofrece una forma eficaz, integrada y gratuita para quien desarrolle en entornos Microsoft de disponer un sistema de control de versiones para un equipo de desarrollo de cualquier tamaño[12].

Team Foundation Server ofrece funciones de seguimiento de elementos de trabajo, además del control del código fuente. Team Foundation Build está incluido y se trata de un sitio web del proyecto de equipo, con generación de informes y administración de proyectos. Además, Team Foundation Server, incluye un almacén de datos donde se guardan las generaciones y las herramientas de pruebas. Cada empresa, según sus necesidades, puede implementar otros servidores. Un servidor de Team Foundation Server se compone de un servidor a nivel de aplicación con servicios web, y un servidor de nivel de datos compuesto por varias bases de datos de SQL Server. El servicio de control de código fuente incorpora las siguientes características: conjunto completo de control de versiones, protecciones de un cambio al mismo tiempo, ramificaciones y recombinaciones optimizadas, aplazamiento de cambios, administración de ramas y coordinación de cambios (con un rol específico de administrador especializado) y directivas de protección[13].

2.1.4.5 Plastic SCM

Ilustración 5: Logotipo de Plastic SCM

Es de las pocas soluciones propietarias de pago basadas en el modelo distribuido, donde claramente gobiernan los sistemas de código abierto gratuitos, y por ello es, que proporcionan una versión gratuita para equipos de desarrollo de máximo 15 miembros. Está desarrollado bajo licencia propia propietaria por la empresa española Codice Software. Sus objetivos son dar un mayor soporte al desarrollo paralelo, creación de ramas, integración de ramas, seguridad y desarrollo distribuido. Está desarrollado bajo la plataforma .NET.

Sus características son: las ramas son creadas como objetos vacíos y solamente cuando un fichero es modificado se asigna una nueva versión a la rama, ramas inteligentes en las cuales el usuario puede definir una jerarquía con soporte de los mecanismos de herencia de etiquetas, conjuntos de cambios, o desde otra rama dinámicamente, explorador de ramas y representante de relaciones, árbol de revisiones en 3D con diferentes tipos de fusiones (regulares, de intervalo, substractivos y de inversión), renombrado y cambios de ubicación de los directorios, seguimiento de fusiones mediante vínculos, seguridad basada en listas de control de acceso con 25 tipos de permisos diferentes, base de datos de *backend* configurable soportando MySQL, SQL Server y Firebird, y espacios de trabajo configurables con mapeo de contenidos[14][15]. Como se puede comprobar, Plastic SCM es un sistema de control muy avanzado (de los que hemos analizado es el más avanzado) y con características únicas, pero su filosofía de pago hace que muchos proyectos colaborativos no apuesten por él, aunque entre su clientela se sitúan compañías como Activision, HP, DHL e Intel y organizaciones como el MIT o la U.S. ARMY.

2.1.4.6 Comparativa

A continuación, se expone una comparativa de las alternativas estudiadas según su precio, modelo de diseño, licencia, ventajas e inconvenientes.

				
CVS	**Subversion**	**Git**	**Visual Studio Foundation Server**	**Plastic SCM**
GPL	Apache/BSD	GPL	Microsoft EULA	Propietaria
Gratis	Gratis	Gratis	Desde 1283,65 €	Gratis hasta 15 usuarios. Desde $279 por usuario extra.
Cliente-servidor	Cliente-servidor	Distribuido	Cliente-servidor	Distribuido
VENTAJAS				
Muchas herramientas disponibles. Popularidad y soporte.	Popularidad y soporte. Gran variedad de clientes. Sencillez e integración con la nube e IDEs. Bloqueo de ficheros. Ahorra ancho de banda.	Popularidad. Pensado para proyectos grandes. Compatibilidad con SVN, CVS y SVK. Ideal para desarrollo no-lineal. Almacenes de información. Diferencias	Integración Visual Studio. Compatibilidad con tecnologías .NET. Generación de informes. Protección de cambios al mismo tiempo. Ramificaciones optimizadas. Aplazamiento de cambios. Gestor de	Soporte empresarial. Soporte a diferentes IDEs. Sólo se asignan a la rama ficheros modificados. Ramas inteligentes con herencia. Visionado de ramas y árbol de revisiones en 3D. Renombrado y cambios de

		basadas en similitudes, no en nombres de ficheros. Notificaciones de cambios.	ramas.	ubicación. Seguimiento de fusiones. BBDD de *backend*.
INCONVENIENTES				
Soporte de renombrado y gestión de ficheros binarios. Obsoleto.	Manejo de orden de cambios de ramas. Función de renombrado no completa.	Menor soporte de herramientas para IDEs y GUIs en sistemas operativos.	No compatible con otros entornos de desarrollo fuera de Visual Studio. Sólo para plataforma Microsoft Windows.	Elevado precio para grupos de desarrollo grandes. Poca popularidad en comunidades colaborativas.

Tabla 1: Comparativa entre sistemas de control de versiones

Por motivos empresariales y de política de Telefónica, el sistema utilizado para las pruebas es Subversion. Para la realización de este proyecto fin de carrera fuera de empresa, el sistema elegido también será Subversion, por las razones aquí expuestas:

- **Integración ya realizada en Telefónica**.
- **Sencillez en su implementación** tanto en la nube (servicios como Dropbox), servidores (que pueden ser de tipo Apache) o localmente en un directorio.
- **Integración con IDEs** de desarrollo utilizados en el desarrollo del proyecto como Eclipse.
- Filosofía de **software libre**, y por lo tanto, con un gran **soporte comunitario**, gran popularidad y gratuidad de la solución.
- **Soporte nativo** del sistema de integración continua **Hudson/Jenkins**, utilizado en Telefónica y parte de la solución de este proyecto fin de carrera.

2.2 Hardware

En este proyecto fin de carrera, el hardware es una pieza esencial, ya que no se realizan pruebas sobre software, sino sobre su comportamiento sobre hardware. Entonces, este apartado se enfocará sobre el software empotrado y el hardware empleado para ello.

Un sistema empotrado, o también conocido con el anglicismo embebido, es un sistema de computación diseñado para realizar una serie de funciones dedicadas a un objetivo en concreto, es decir, no están diseñados para propósito general como los ordenadores personales. El software se suele memorizar dentro de un artefacto hardware compuesto por una arquitectura de microcontrolador, FPGA o microprocesador, entre otras alternativas. Por lo general, este tipo de sistemas son muy fiables ya que están testeados totalmente para cumplir su función sin errores. Esta fiabilidad es más trascendente en los sistemas empotrados de tiempo real o críticos, donde no se pueden tolerar errores de ningún tipo.

Existen diferentes posibilidades, desde microcontroladores con instrucciones básicas ejecutadas en bucle hasta arquitecturas propietarias con sistemas operativos empotrados de diferentes orígenes (por ejemplo: Windows Embebbed en microprocesadores, o MicroLinux en FPGAs). En la elección de arquitectura hardware y la complejidad del software correspondiente influyen factores como el rendimiento, estabilidad, coste, consumo eléctrico, tamaño y miniaturización, etc.

La programación parte desde el lenguaje ensamblador del microcontrolador o microprocesador en concreto, donde se puede gestionar el hardware a medida optimizando los recursos aprovechando sus características. Aunque, lo más frecuente, por simplicidad y abstracción, es utilizar compiladores específicos, al igual que en los ordenadores personales. Los lenguajes de alto nivel posibles son C o C++ o alguna variante personalizada en la mayoría de los casos, aunque, en algunos casos contados y si la optimización no es un factor crítico, pueden utilizarse lenguajes interpretados como Java.

Los componentes de un sistema empotrado pueden ser muy variados dependiendo de su función final, por lo que sólo se nombran algunos básicos (aunque pueden no estar todos en un sistema empotrado final):

- **Microprocesador, microcontrolador, DSP, etc.:** es la CPU o unidad de cómputo del sistema. Puede incluir arquitectura específica según requisitos.

- **Hardware de comunicación:** existen multitud de interfaces muy usadas en la electrónica industrial y en la informática y que además pueden incorporar su propia electrónica. Como ejemplo podemos citar RS-232, RS-485, Serial o USB, I2C, ModBus o Ethernet, aparte de soluciones inalámbricas como GSM, GPRS, WIFI, etc.

- **Hardware de presentación:** suele ser una pantalla gráfica normal o táctil, LCD, panel BCD, conjunto de LEDs, etc.

- **Hardware de E/S:** módulo empleado para digitalizar señales analógicas procedentes de sensores y proporcionar salidas de señales digitales. Suelen ser conectores de pines y entre sus usos más básicos destacan activar diodos LED o emitir estados de un conmutador.

- **Actuadores:** elementos electrónicos que el sistema puede controlar al estilo de un motor o un relé, entre otros. Lo más habitual suele ser la inclusión de una salida de señal PWM (*Pulse Wide Mode*) para control de frecuencias, intensidades, etc., en las salidas de señales digitales.

- **Hardware de reloj:** se encarga de generar las diferentes señales de reloj a partir de un único oscilador principal. Este componente es fundamental para la sincronización, ya que introduce la frecuencia necesaria para la estabilidad de las comunicaciones y cumplir con el consumo de corriente requerido. Mayormente se utilizan los basados en resonador de cristal de cuarzo por ser idóneos para las características mencionadas, aunque existen diferentes tipos más eficaces en diferentes aspectos.

- **Hardware de gestión de energía:** controla y mantiene las diferentes tensiones y corrientes necesarias para alimentar los diferentes circuitos del sistema, impidiendo que se quemen elementos por sobrealimentación o malfuncionamientos por falta de ella. Se componen de conversores, condensadores, y demás elementos electrónicos de filtrado. En circuitos alimentados por batería también se encargan de su gestión.

Las características de los sistemas empotrados son las siguientes:

- **Fiabilidad y seguridad:** diseños cerrados y testeados.

- **Eficiencia:** sistemas de tiempo real, recursos limitados pero adecuados sin derroches, gran optimización mediante ensamblador o lenguaje C.

- **Interacción con dispositivos físicos:** a través de hardware de E/S y comunicación.

- **Bajo consumo:** sistemas alimentados con pilas o baterías.

- **Bajo peso y tamaño:** útil en sistemas portátiles o poco visibles.

- **Bajo precio:** aplicable a electrónica de consumo y mercados muy competitivos con grandes alternativas.

Las implementaciones de un sistema empotrado pueden abarcar desde dispositivos programables (FPGAs), circuitos a medida (ASICs) con microcontroladores, *Gate Arrays* o incluso PCs completos en *SOCs*[16]. Por el interés del proyecto, sólo abordaremos los microcontroladores, microprocesadores y FPGAs, ya que los *Gate Arrays* y ASICs son formas de realizar circuitos a medida y poseen poca relación con el marco, objetivos y contexto del proyecto realizado en Telefónica.

2.2.1 Microcontroladores

Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto por varios bloques funcionales, los cuales cumplen una tarea específica. Los microcontroladores suelen basarse en la arquitectura Von Harvard, y por lo tanto, poseen memoria y E/S propias aparte de la unidad central de proceso. Al ser fabricados, la memoria EEPROM o equivalente no posee datos, por lo que el programa se graba posteriormente a través de una codificación hexadecimal del propio programa.

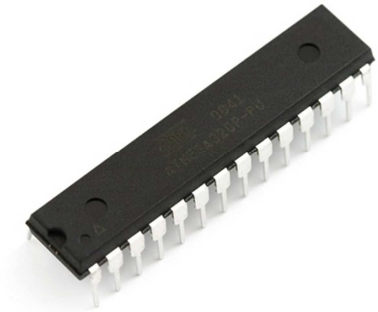


Ilustración 6: Microcontrolador ATMEGA328-PU de Atmel

Por lo general, tienen la capacidad de mantener la funcionalidad a la espera de un evento como pulsar un botón o de otra interrupción, el consumo de energía durante el sueño puede ser sólo de nanovatios, lo cual los hace muy recomendables para aplicaciones con baterías de larga duración. También pueden actuar como procesadores digitales de señal. Otra ventaja es su costo económico debido a su potencia contenida adecuada para las necesidades de su diseño.

Los microcontroladores representan una gran mayoría del mercado de chips, ya que multitud de elementos con sistemas empotrados como taxímetros, electrodomésticos, automóviles, teléfonos, módems, etc., pueden incorporar un gran número de ellos. En cambio, un ordenador personal incluye un microprocesador.

Podemos agrupar los tipos de microcontroladores según sus prestaciones o según su tecnología. Según sus prestaciones, tenemos gama baja de 4, 8 y 16 bits dedicados a tareas de control, gama media de 16 y 32 bits dedicados a tareas de control con cierto grado de procesamiento y acompañados de chips externos y gama alta de 32, 64 y 128 bits dedicados a videoconsolas, ordenadores y que casi en su totalidad son microprocesadores con circuitería periférica y memoria. Según la tecnología, se pueden clasificar según la alimentación en 5v, 3.3v, 2.5v o 1.5v, según el consumo desde microvatios a decenas de vatios o según la frecuencia desde KHz a GHz[17][18].

Entre los microcontroladores más utilizados es imprescindible destacar a Atmel con sus AVR, Microchip con PIC y ARM Holdings con ARM.

2.2.2 Microprocesadores

Ilustración 7: Microprocesador de arquitectura UltraSparc

Los microprocesadores son circuitos integrados complejos formados por millones de componentes electrónicos. Están constituidos por registros, una unidad de control, una unidad aritmético-lógica y una unidad de cálculo en coma flotante (antiguamente denominada co-procesador matemático), aparte de memorias propias de agilización de la ejecución conocidas como memorias caché, en varios niveles. Siguen la arquitectura Von Neumann, por lo que necesita una memoria principal y un sistema de entrada y salida externo, conectados a lo que se denomina placa base o *motherboard* junto al zócalo del microprocesador y comunicados por un bus. Son utilizados en los ordenadores personales.

Existe bastante variedad de arquitecturas, con diferentes niveles de rendimiento, costes, temperatura, alimentación, etc., para diversos usos. Si bien, los más económicos y de menor consumo se suelen utilizar en sistema empotrados (teléfonos inteligentes, microrordenadores al estilo Raspberry Pi o Pandaboard), para tareas simples que requieran una gran duración de energía o que no requieran de demasiado cómputo se siguen prefiriendo los microcontroladores por su sencillez, ya que los microprocesadores no guardan ningún programa, sino que acceden a él en memoria principal, por lo que necesitan un sistema operativo que gestione el conjunto del hardware, con la complejidad que conlleva. En otras palabras, suelen utilizarse si se requiere alta capacidad de procesado.

Unos ejemplos de microprocesadores son las arquitecturas x86 de Intel y AMD, la arquitectura PowerPC de IBM o la arquitectura MIPS.

2.2.3 Microprocesadores vs. Microcontroladores

La diferencia entre microcontrolador y microprocesador es que el microprocesador al basarse en la arquitectura Von Neumann, se utiliza memoria principal externa al microchip y sistemas de entrada y salida, por lo que entonces, el microprocesador ahorra un buen número de líneas de E/S, al contrario que en la arquitectura Harvard, usada en la mayoría de los microcontroladores, donde existen problemas por el número de líneas utilizadas, ya que el bus de datos es interno, estando la memoria incluida en el microchip, y no existe un sistema de E/S, con lo cual, deben incluirse pines para cada E/S que se desea proporcionar.

En el caso de los microcontroladores, existen dos tipos de memoria bien definidas: memoria de datos (algún tipo de *SRAM*) y memoria de programas (*ROM*, *PROM*, *EEPROM*, *flash* o similar), en este caso la organización de memoria es diferentes por la existencia de circuitos distintos por memoria, además de que normalmente la memoria está segregada y el acceso depende de las instrucciones del procesador.

Por el contrario, en los microprocesadores, el programa se guarda en memoria principal junto con los datos, y ésta posee un espacio de direcciones que se divide en segmentos, de los cuales típicamente tendremos código (programa), datos y pila. Es por ello, que podemos hablar de memoria en los microprocesadores como un todo, aunque existan diferentes dispositivos físicos.

El formato unitario, en un sólo microchip, y el grado de independencia del microcontrolador hace que sea predilecto en los sistemas empotrados sencillos, por su consumo, potencia ajustada, fácil programación, simplicidad, coste y porque incluye todo lo necesario para su funcionamiento en su arquitectura: *EEPROM* de programa, E/S, temporizadores, *UARTs* o buses serie, y además sólo hace falta un cristal de sincronización que le arroje una señal de reloj para hacerlo funcionar, además de la alimentación[18].

Los criterios de selección entre microprocesador o microcontrolador son muy variados: requisitos y coste, herramientas de desarrollo, experiencia y soporte, compatibilidad, disponibilidad y segundas fuentes, repercusión del tiempo de desarrollo sobre los beneficios, vida media de un producto y obsolescencia de la tecnología.

2.2.4 FPGAs

FPGA, cuyas siglas significan *Field Programmable Gate Array*, es un dispositivo semiconductor reprogramable que permite implementar diversos circuitos digitales en él. Generalmente están basadas en memorias RAM. Existen aplicaciones en las cuales utilizar un microcontrolador no es suficiente o usar una FPGA supone un costo comparable, como por ejemplo al decodificar y codificar MPEG. Actualmente hay modelos que incluyen elementos adicionales en el mismo chip, como el modelo Virtex 4 de Xilinx que posee uno o más procesadores PowerPC empotrados en el chip. Las posibilidades de programación abarcan desde una puerta lógica a un sistema complejo en un chip[19].

La arquitectura de una FPGA posee al menos 3 bloques:

- ***Configurable-Logic Blocks***: implementan los circuitos lógicos diseñados.
- ***Input-Output Block*****:** en este bloque se conectan las configuraciones internas con pines de entrada y salida.
- ***Digital Clock Managers*****:** gestionan la entrega de señales de reloj a toda la FPGA.

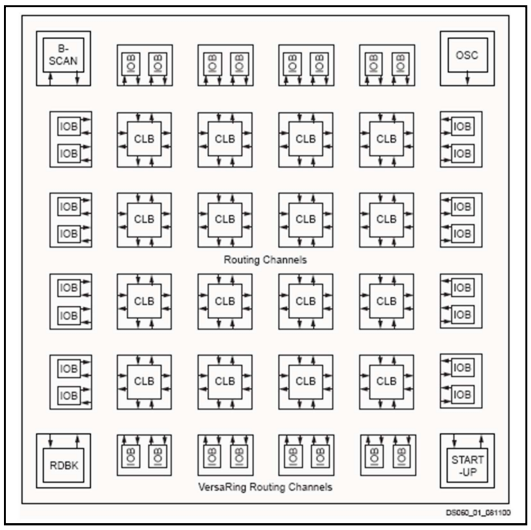


Ilustración 8: Arquitectura básica de una FPGA de Xilinx Corp.

El elemento esencial de los circuitos programables es el *slice*, ya que, es donde se pueden implementar tanto circuitos combinacionales como secuenciales. Entre los elementos básicos utilizados destacan *buffer* tri-estado, *look-up table*, *flip-flop* (sobre todo el tipo D), multiplexor y *demux*/decodificador. Una *slice* es, en esencia, un bloque de *look-up tables* enlazadas con una salida que puede ser registrada por un *flip-flop* o directa, pudiendo implementarse diversas funciones lógicas. Para implementar funciones combinacionales, se programan las tablas de verdad y se usa la salida directa. Por el contrario, para funciones secuenciales, se utiliza el *flip-flop* y se realimenta la salida secuencial en una de las entradas si es necesario.

La programación no consiste en utilizar funciones y variables como en los microcontroladores y microprocesadores, sino que consiste en describir el hardware que implementará la FPGA. Por consiguiente, la tarea del programador es definir la función lógica que realizará cada uno de los bloques configurables (*Configurable-Logic Blocks*), seleccionar el modo de trabajo de cada *Input-Output Block* e interconectarlos. A priori, puede parecer demasiado complejo, pero el programador cuenta con entornos de desarrollo especializados en el modelo concreto de FPGA a utilizar, proporcionados por los fabricantes.

Un diseño puede ser de tipo esquemático o de tipo lenguaje de programación. Estos lenguajes de programación no son los típicos ya que son especiales por las diferencias de programación entre FPGAs y el resto de procesadores y controladores. Se suelen conocer como HDL (*Hardware Description Language*) y existen diferentes variedades: VHDL, Verilog, ABEL, etc. Existen diferentes niveles de abstracción del diseño, desde niveles funcionales superiores a niveles de componentes hardware básicos, además de acercamientos a programación gráfica de alto nivel por parte de algunos fabricantes. Bajo licencia GPL se encuentra disponibles códigos, denominados *cores*, que programan microprocesadores, microcontroladores, filtros, módulos de comunicaciones, y memorias, entre otros.

Las ventajas de las FPGAs respecto a otras formas de implementar circuitos es que son dispositivos configurables, poseen bajo costo en comparación los ASICs, sirven como testeo de circuitos definitivos y además se ejecutan más rápido que en otros dispositivos programables. Al ser circuitos digitales, la ejecución de cada bloque es en paralelo, imposible de emular en un microcontrolador. En cuanto a las desventajas, al estar basadas en RAM pierden su configuración al suprimir la energía, aunque existen soluciones para ello. Además, poseen retardos de propagación mayores a los existentes en los ASIC con procesadores de velocidades aproximadas al GHz. Actualmente sus aplicaciones más comunes consisten en DSPs, radio definido por software, sistemas aeroespaciales y de defensa, prototipado de ASICs, sistemas de imágenes para medicina, sistemas de visión de computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, etc[20].

Entre sus fabricantes destacamos a Xilinx, Altera, Atmel, Actel, Lattice Semiconductor, Cypress Semiconductor, Achronix Semiconductor y QuickLogic.

2.2.5 Hardware abierto

Se conoce como *open hardware* o hardware abierto a dispositivos de hardware cuyas especificaciones y diagramas esquemáticos son de acceso público. Siguen la misma filosofía que el software libre, existiendo multitud de licencias: Free Model Foundry, ESA Sparc, Free-IP Project, GNUBook, etc. Suelen ser sistemas empotrados, aunque a veces albergan procesadores complejos como la arquitectura UltraSparc, o incluso sistemas de videojuegos.

Actualmente, las empresas pueden comercializar un diseño libre implementándolo siempre que se cumpla con la premisa de mantenerlo libre, de esta manera, pueden ahorrar costes y tiempos de diseño, ya que cuentan con un equipo de desarrolladores repartidos por el mundo. La mayoría de empresas poseen un miedo a no patentar para exprimir los posibles beneficios, por esta razón, los desarrolladores suelen encontrarse en instituciones universitarias y públicas, donde la motivación investigadora es evidente además de la gran afluencia de ideas.

La utilización de hardware abierto nos provee de múltiples ventajas: protege la soberanía de cada país para evitar la dependencia de desarrollo extranjero, fomenta el hardware de calidad, abierto y económico, permite la reutilización y adaptación de diseños a nivel mundial de forma colaborativa, posibilita a las empresas ahorrar tiempo y dinero, promulga comunidades de desarrollo crecientes y participativas y evita la alianza *trusted computing* y la gestión digital de derechos (DRM).

A primera vista, el open hardware puede ser un éxito al igual que su homólogo el software de código abierto, pero el mundo del hardware es totalmente diferente y funciona bajo otros mecanismos, lo que aporta unas desventajas muy críticas a este tipo de modelo de desarrollo: la compartición de un diseño único depende de la facilidad de su reproducción, además de que tiene asociado un coste de construcción y verificación, la disponibilidad de los componentes es un problema frecuente según el país y la edad del diseño pudiendo ocasionar rediseños costosos, se debe evitar vulnerar patentes siendo el mundo del hardware una industria plagada de ellas, y además, y por último, la duplicación de hardware requiere de infraestructura de diseño, simulación, producción e implementación, justamente al contrario que el software libre que cualquiera puede desarrollar en un simple ordenador mediano[21].

2.2.6 Soluciones de computación open hardware

Existen diversas soluciones para controlar o procesar información mediante hardware abierto de costo moderado y fácil de entender y utilizar. A continuación, analizamos las más relevantes haciendo hincapié en Arduino, que es la solución utilizada por el proyecto y Telefónica.

2.2.6.1 Arduino

Ilustración 9: Logotipo de Arduino

Arduino es una plataforma de hardware libre diseñada para acercar la electrónica fácil a cualquier persona, en cualquier tipo de proyecto multidisciplinar. Básicamente su arquitectura se compone de un microcontrolador con su electrónica complementaria para su correcto funcionamiento. Arduino puede ser utilizado para diseñar y fabricar dispositivos autónomos ligeros, de tamaño y de coste moderado (alrededor de 20 dólares el Arduino Uno), y con múltiples posibilidades de conectividad y ampliación. El objetivo final es el acercamiento de la electrónica a la mayoría de la gente, aunque ésta no haya tocado un componente electrónico, realizando una plataforma intuitiva, con mucho soporte y tutoriales y ejemplos muy básicos, promocionando la idea del "hazlo tú mismo".

Entre sus características, podemos destacar las siguientes[22]:

- **E/S:** dependiendo del modelo de placa, consta de entradas analógicas y entradas/salidas digitales operando a 5 voltios y 40 mA como máximo. Soporta PWM en diversos pines. Los pines 0 y 1 suelen interferir con la comunicación USB. Es posible cambiar el voltaje de los pines utilizando un pin especial Aref y código de bajo nivel.

- **Microcontroladores:** según el modelo de placa éstos pueden variar, siempre dentro de los microcontroladores ATMega de Atmel. Los más comunes son el ATMega168, el ATMega328 y ATMega1280. Todos funcionan con 5 voltios aproximadamente y una intensidad de 40 mA. Los pines de entrada y salida varían: 14 digitales con 6 PWM y 6 analógicos sólo entrada en el ATMega168 y ATMega328 y 54 digitales con 14 PWM y 16 analógicos sólo entrada en el ATMega1280. La memoria *flash* también varía: 16KB (2 de *bootloader*), 32KB (2 de *bootloader*) y 128KB (4 de *bootloader*) en los ATMega168, ATMega328 y ATMega1280 respectivamente. Además, poseen dos tipos de memoria: SRAM, de 1KB para ATMega168, 2KB para ATMega328 y 4KB para ATMega1280, donde se almacenan los datos variables, y EEPROM, de 512 bytes para ATMega168, 1KB para ATMega328 y 4KB para ATMega1280, donde se almacenan las instrucciones del programa. Como se puede comprobar, el ATMega1280 es el más completo, aunque todos, los 3 microcontroladores presentados, funcionan a 16 MHz. También es posible encontrar otros microcontroladores menos comunes como el ATMega2560 o versiones especiales de los anteriores.

- **Entorno de desarrollo:** la plataforma ofrece su propio IDE o entorno de desarrollo llamado Arduino IDE, desarrollado en Java y disponible para plataformas Windows, GNU/Linux y OS X, aparte de la versión con el código fuente, ya que es de código abierto. Arduino IDE proporciona todas las herramientas para la creación de ficheros de programa, llamados *sketches*, y su compilación y carga en la placa mediante las herramientas AVR de Atmel de una forma fácil e intuitiva, también combina herramienta de edición de código: búsqueda, impresión, resaltado y colores de texto, inclusión de librerías, etc., y facilita ejemplos y librerías para usuarios básicos. También posee funciones avanzadas como la carga de un *bootloader*, el desarrollo de librerías o la inclusión de ficheros auxiliares en un *sketch*.

- **Programación:** en el IDE oficial, el lenguaje de programación es C o C++ con una API añadida propia para gestionar las características de Arduino, similar a Processing. Los *sketches* suelen implementarse en C soportando toda su API estándar ANSI C, mientras las librerías son objetos creados a partir de clases realizadas con C++ (de la API de C++ sólo se soportan algunas funciones). Sin embargo, es posible utilizar otros lenguajes de programación y aplicaciones populares: Java, ActionScript, Processing, Python, Ruby, Objective-C, Matlab, Perl, Visual Basic .NET, VBScript, Gambas, SuperCollider, etc., esto es posible porque Arduino se comunica mediante transmisión de datos en serie, algo soportado por la mayoría de los lenguajes citados, o en caso contrario puede ser soportado con una herramienta o librería de terceros. Por supuesto, también permite la programación mediante ensamblador de Atmel.

- **API y librerías:** la API proporciona una extensión de C/C++ con funciones para manejar pines y características del microcontrolador, pero con un nivel de abstracción muy elevado. Las librerías o bibliotecas son extensiones de funcionalidad que podemos añadir al *sketch* al igual que en la programación de ordenador común, el IDE incluye varias básicas: Serial (comunicación a través de puerto serie), EEPROM (manejo de memoria de programa), Ethernet (conexión a través del *shield* Arduino Ethernet), Firmata (comunicación con aplicaciones de ordenador por serie), LiquidCrystal (control de LCDs tipo Hitachi HD44780), Servo (control de servo motores), SoftwareSerial, Stepper (control para motores paso a paso) y Wire (comunicación I2C). Cualquiera puede crear sus propias librerías de forma sencilla, con conocimientos básicos de paradigma de objetos y C++, incluyendo la cabecera Arduino.h en su código.

- **Extensiones hardware:** Arduino se puede extender enchufándole escudos o *shields* que le proporcionan hardware extra para objetos más complejos. Suelen venir acompañados de librerías propias para facilitar y gestionar el manejo de dicho hardware. Los *shields* pueden estar fabricados por terceras personas, algunos reciben la certificación oficial de Arduino. Entre los diferentes *shields* podemos encontrar bastante variedad, por citar algunos: Ethernet Shield, WIFI Shield, Telefónica I+D GSM/GPRS Shield, SoundWave Shield, etc.

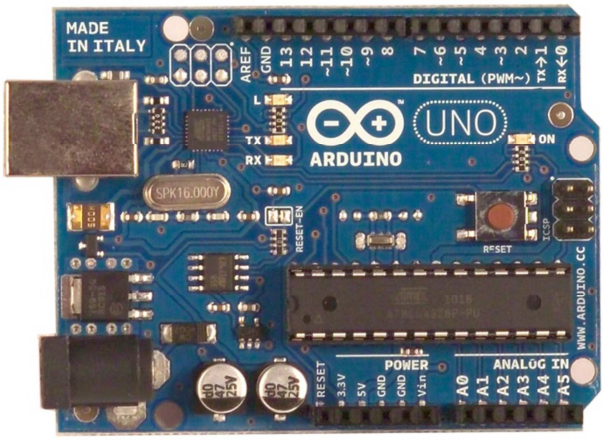


Ilustración 10: Placa Arduino UNO

Existen diferentes modelos de Arduino, aparte de los fabricados por terceras personas, cada uno con características propias, y diseñados para usos y situaciones generales o específicas, repasamos los principales:

- **Arduino Uno:** la versión básica y más sencilla de Arduino, controlado por el ATMega328P (16 MHz, 32 KB *flash*, 2 KB *SRAM*, 1 KB *EEPROM* y 5 V). Dispone de 14 pines digitales E/S (6 con PWM) y 6 analógicos de entrada, conexión serial, SPI, LED directo e I2C, botón de *reset*, conexión a PC mediante USB de tipo B controlado por un ATMega16U, conexión ICSP para programar directamente el microcontrolador principal y el microcontrolador dedicado al USB mediante un programador y LEDs indicativos de Power y TX/RX[23].

- **Arduino Leonardo:** la versión más reciente de Arduino que sustituirá al Arduino Uno. La estructura en cuanto a conexiones es la misma, la diferencia radica en la utilización del microcontrolador ATMega32U4 (16 MHz, 32 KB *flash*, 2,5 KB *SRAM*, 1 KB *EEPROM* y 5 V), que es capaz también de gestionar el puerto USB y la traducción FTDI a serial, por lo que sólo es necesario un microcontrolador en placa. La conexión a PC se moderniza empleando el estándar de facto MicroUSB[24].

- **Arduino Mega 2560:** versión profesional destinada al desarrollo de aplicaciones complejas de mayor tamaño y que requieran un gran número de interacciones por conexiones con el microcontrolador. Esto es posible al incorporar el microcontrolador ATMega2560 (16 MHz, 256 KB *flash*, 8 KB *SRAM*, 4 KB *EEPROM* y 5 V), que posee mayor capacidad que el resto de modelos. Se proporcionan las mismas posibilidades de conexión, añadiendo 15 entradas analógicas y 54 pines de E/S digitales (14 admiten PWM), 6 entradas de interrupción externas para el microcontrolador y 3 conexiones seriales. Como se puede apreciar, es bastante más completo. La conexión a PC se realiza por USB de tipo B y requiere del microcontrolador para su gestión al igual que el Arduino Uno (en este caso, el ATMega16U2)[25].

- **Arduino LilyPad:** versión especial de Arduino realizada para su colocación en materiales textiles posibilitando la creación de dispositivos e-textiles. Está diseñado para soportar alimentación, sensores y actuadores conectados por hilo conductivo. Existen dos variantes con dos microcontroladores: ATMega168V o ATMega328V (ambos a 8 MHz, 16 KB *flash*, 1 KB *SRAM*, 512 bytes *EEPROM* y 2,7-5,5V). Como se puede ver, es menos potente que el resto de modelos. Tiene forma redonda y las conexiones en disposición de flor, éstas son 6 entradas analógicas y 14 E/S digitales (6 permiten PWM). La conexión con el PC sólo se puede realizar mediante ICSP, lo que requiere programador[26].

- **Arduino Mega ADK:** versión profesional posterior al Arduino Mega 2560, enfocada, además, a su uso con el sistema operativo Android de Google mediante la mediación con un dispositivo externo. El microcontrolador es el ATMega2560 ya explicado. Las conexiones y posibilidades son las mismas que el Arduino Mega 2560, incluyendo número de pines, microcontrolador para la conexión USB, protocolos, etc. La diferencia fundamental es que, aparte de la conexión USB tipo B para el PC con su microcontrolador dedicado, dispone de otra conexión USB tipo A en formato hembra, con un chip MAX3421e IC, destinado para la conexión de un dispositivo Android en ella[27].

Existen otros modelos de Arduino para necesidades aún más específicas pero con conexiones y potencia similares, enumeramos algunos: Arduino Duemilanove como versión anterior al Arduino Uno, Arduino Fio para aplicaciones inalámbricas con soporte de conexión XBee y de baterías de polímero de litio, Arduino Ethernet para aplicaciones conectadas con soporte de conexión RJ45, Arduino Pro para instalaciones semipermanentes en objetos o exhibiciones con soporte de batería de polímero de litio y programación mediante ICSP, Arduino Bluetooth para comunicaciones con dispositivos Bluetooth (excepto *headsets* y dispositivos de audio), Arduino Nano como el Arduino más pequeño existente para espacios muy reducidos incorporando la misma potencia que un Arduino Duemilanove, entre otros[28].

No contamos con que, al ser open hardware, cada uno puede construir su propia plataforma basada en Arduino con sus componentes preferidos y adecuados a sus necesidades. Un buen número de distribuidores y tiendas ofrecen kits de trabajo que incluyen componentes electrónicos para desarrollar cualquier idea, además de una placa Arduino Uno. Se han desarrollado un buen número de sensores de todo tipo y actuadores con soporte para Arduino incluyendo las librerías necesarias (cámaras, motores, sensores de gas, luz, presión, temperatura, etc.). Existen multitud de empresas y universidades que lo utilizan como plataforma base de desarrollo hardware, como Telefónica, aparte de la gran comunidad que posee detrás, lo que proporciona un gran soporte y multitud de soluciones e ideas ya implementadas por desarrolladores de todo el mundo.

2.2.6.2 OpenPICUS (FlyPort)

Ilustración 11: Logotipo de OpenPICUS

OpenPICUS es una compañía de open hardware que produce productos de libre diseño basados en microcontroladores de la familia PIC. La plataforma producida se denomina FlyPort. Sus productos se caracterizan por tener un enfoque más orientado al mercado real, aunque también pueden ser destinados para la investigación y aficionados (es posible que a Arduino le falte un enfoque un poco más comercial como poseen los productos de OpenPICUS), y como no, como plataforma libre base para el desarrollo de productos por terceros.

FlyPort posee ventajas similares a Arduino. Entre estas ventajas encontramos precios económicos, sencillez de diseño y programación, tamaños moderados, múltiples posibilidades de expansión y funcionamiento autónomo. Aun así, se encuentran diferencias en la orientación de ambos: Arduino es de propósito general y FlyPort está destinado a la comunicación con internet asimilando el concepto del internet de las cosas, en otras palabras, todos los FlyPort son módulos hardware con conexión a internet, mientras que un Arduino básico requiere de un escudo para ello. Además, otra diferencia es el uso promovido por OpenPICUS de FlyPort como servidor web empotrado para almacenar y mostrar páginas HTML personalizadas con los datos de sensores o con interactividad para activar/desactivar actuadores[29].

A continuación, se describen los productos OpenPICUS y su plataforma FlyPort:

- **FlyPort:** es una plataforma basada en un microcontrolador PIC de 16 bits y un transceptor de comunicación. Actualmente existen dos versiones: FlyPort Wi-Fi y FlyPort Ethernet, además se está desarrollando una versión con un módulo GPRS. FlyPort está muy orientado al concepto del internet de las cosas, facilitando la conexión y el envío/recepción de información a través de internet, ya sea a través de servicios TCP/IP o un servidor web personalizable. Las conexiones de pines son personalizables mediante software. A continuación, se repasan las características de las dos versiones disponibles:

    - **FlyPort Wi-Fi:** incorpora un módulo integrado Wi-Fi compatible con el estándar 802.11b y soporta HTTP, TCP, UDP, SNTP, FTP y SMTP, puede funcionar en modo ad hoc o en modo infraestructura. El microcontrolador es el PIC 24FJ256 de 16 bits, 8 MHz y 16 MIPS, con 256 KB de *flash* para contener un servidor web empotrado, la pila de comunicación o la aplicación personalizada, soporta modo bajo consumo y de hibernación, proporciona conexiones UART, I2C y SPI y pines de E/S analógicos y digitales que pueden funcionar en diversos modos (I2C, interrupción, etc.)[30], soporta USB *On-The-Go* y USB como dispositivo. Existen dos variantes en el mercado: con una antena PCB o conector uFL[31].

- **FlyPort Ethernet:** similar a la versión Wi-Fi, incorpora un microcontrolador similar PIC familia 24FJ256 con las mismas posibilidades de conexión pero con una memoria *flash* de 16 Mbit. Además, la placa añade un módulo Ethernet BASE-T 10/100. Existen dos variantes: con conector RJ45 o sin él (añadido con una expansión). Soporta actualización del software a través de internet[32].

Ilustración 12: Placa FlyPort Wi-Fi

- **Nest Expansion Boards:** expansiones para el desarrollo personalizado de aplicaciones compatibles con FlyPort. Existen multitud de plataformas ya implementadas y distribuidas comercialmente[33]:

  - **Lighting Nest:** destinado para aplicaciones de control de iluminación a través de internet, incluye 4 relés, 2 entradas de tipo optoacoplador y 1 triodo de corriente alterna.

  - **Serial Nest:** destinado para comunicaciones a través de interfaces RS-232 y RS-485.

  - **EnOcean Nest:** proporciona conectividad EnOcean para aplicaciones de domótica y automatismo del hogar. EnOcean es un estándar de comunicación inalámbrica entre dispositivos del mundo de la domótica.

  - **Music Nest:** diseñado para aplicaciones de audio estéreo sobre IP. Incorpora un códec estéreo VLSI1053 y un slot MicroSD para almacenamiento local formateado en FAT32.

  - **Arduino Shield Nest:** expansión que posibilita el uso de escudos de Arduino con FlyPort. Es relevante recalcar que es compatible con el esquema eléctrico y mecánico del Arduino estándar, pero no hay ningún microcontrolador compatible con Arduino, ya que FlyPort actúa como microcontrolador.

  - **Otros:** existen extensiones destinadas a conectividad básica y prototipos como USB Nest, Proto Nest y MiniUSB *programmer*.

FlyPort puede funcionar bajo un ligero y sencillo sistema operativo, FreeRTOS, dedicado completamente a implementar una pila TCP/IP con un servidor web empotrado. También, se puede implementar software personalizado mediante OpenPICUS IDE, que es software libre y gratuito. OpenPICUS IDE permite escribir el código necesario, compilarlo y cargarlo en la placa FlyPort, además, de ser un gestor de las páginas webs del servidor web empotrado. Proporciona una manera de configurar los parámetros de red sencilla completando unos pocos formularios, también permite cambiar los parámetros en tiempo de ejecución. El editor de código contiene herramientas de autocompletado y resaltado de texto. Integra monitor serie para la depuración y el envío directo de comandos.

El servidor web soporta páginas HTML dinámicas mediante jQuery, también puede funcionar como cliente FTP, cliente de correo electrónico o como cliente o servidor TCP/UDP. La programación está basada en el estándar ANSI C, que al igual que Arduino, incorpora su propia para el manejo del PIC y los protocolos de red soportados, además de librerías externas[34].

2.2.6.3 Raspberry Pi

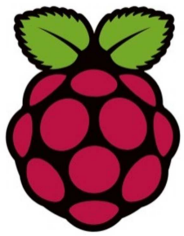


Ilustración 13: Logotipo de Raspberry Pi

Raspberry Pi es un producto muy reciente cuyo lema de entrada es: "*An ARM GNU/Linux box for $25*.". Se trata de una placa open hardware basada en un microprocesador, por lo que la arquitectura es diferente ya que requiere de chips externos de memoria y E/S. En otras palabras, es un microrordenador muy económico con GNU/Linux, aunque se le pueden instalar otros sistemas operativos, y unas posibilidades de comunicación sorprendentes para su potencia y tamaño. El objetivo del fabricante, la Fundación Raspberry Pi, una asociación caritativa inglesa, es "promover el estudio de las ciencias de la computación y temas relacionados, sobre todo a nivel escolar, y para recuperar la diversión de aprender computación". Fundamentalmente se promueve el aprendizaje de los lenguajes de programación Python, BBC BASIC, C y Perl.

En términos de hardware, las primeras versiones beta integraban un microcontrolador ATMega644, sus esquemas son de dominio público. Posteriormente, se apostó por un microprocesador ARM y una arquitectura más parecida a un teléfono móvil inteligente actual, se consiguió ejecutar con éxito el escritorio LXDE en el sistema operativo Debian GNU/Linux, junto con Quake 3 en resolución 1080p y video Full HD con códec H.264. También se implementó una versión de RISC OS 5.

Las placas definitivas incorporan un SOC (*System-On-a-Chip*) Broadcom BCM2835 que integra CPU, GPU, DSP y SDRAM. La CPU consiste en un microprocesador ARM1176JZF-S a 700 MHz, la GPU es un chip Broadcom VideoCore IV con soporte OpenGL ES 2.0 y decodificador dedicado H.264 que proporciona una resolución de 1080p a 30 cuadros por segundo. La memoria SDRAM es compartida con la GPU y posee 256 MiB. Se alimenta con 5 V vía MicroUSB o GPIO *header*. Las conexiones dependen del modelo, ya que existen dos: modelo A y modelo B, de 25 dólares y 35 dólares respectivamente. Ambos modelos incorporan puertos USB 2.0 (1 en el modelo A y 2 en el modelo B), salida RCA y HDMI de video, salida *jack* 3.5mm y HDMI de audio, almacenamiento SD/MMC/SDIO (actúa como almacenamiento permanente, ya que no se incluye disco duro) y pines GPIO, SPI, I2C y UART. El modelo B incorpora exclusivamente un puerto RJ45 con soporte Ethernet 10/100 integrado[35].

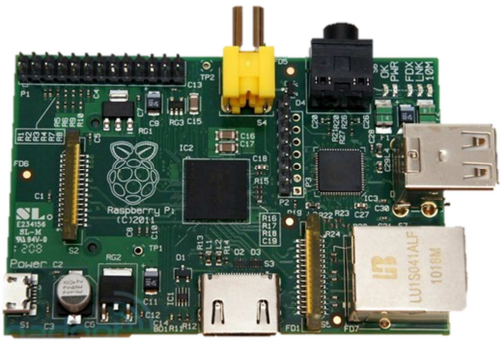


Ilustración 14: Placa Raspberry Pi

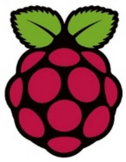
En cuanto al software, los sistemas operativos soportados oficialmente son Debian GNU/Linux, Raspbian, QtonPi y Arch Linux ARM, todos son distribuciones de GNU/Linux en sus versiones compiladas para arquitectura ARM. RISC OS dejó de ser soportado. Fuera de lo oficial, se han portado sistemas operativos como Android, Chromium OS, XMBC, Fedora *Remix for Raspberry*, etc. En cuanto a plataformas Microsoft, se ha logrado ejecutar Windows 7 x86 mediante virtualización con Citrix XenDesktop 5.6, Windows 8 no será compatible porque la CPU presenta una arquitectura ARM considerada antigua por Microsoft[36].

Existe competencia en este campo de los microrordenadores económicos, ejemplos muy parecidos en arquitectura son BeagleBoard o Pandaboard, aunque tienen la desventaja de ser un poco superiores en precio (alrededor de 100$) y no disponer de la comunidad y la atención tan grande que está creando Raspberry Pi. Por parte de las empresas dedicadas a la computación, Intel ha presentado su NUC, una alternativa que integra un microprocesador Intel Core i3/i5 Sandy Bridge, Wi-Fi, Bluetooth, Thunderbolt, USB 3.0 y HDMI[37], sin duda superior en hardware y posibilidades, pero seguramente más caro y sin la filosofía open hardware, factores fundamentales que hacen que Raspberry Pi se esté imponiendo en este mercado.

Las posibilidades comerciales para diferentes usos no se encuentran todavía explotadas, ya que el producto es muy reciente. Factores como las filosofías open hardware y software libre, el coste muy económico, las posibilidades enormes en un tamaño pequeño, la compatibilidad con GNU/Linux y toda su comunidad y la orientación a la formación en universidades, conllevan que su utilización pase por proyectos amateurs realizados por usuarios de nivel medio-avanzado o avanzado, simple curiosidad o para formación e investigación.

2.2.6.4 Comparativa

A continuación, se expone una comparativa de las alternativas estudiadas según su precio, arquitectura hardware, programación, software, especificaciones, ventajas e inconvenientes.

		
Arduino	**OpenPICUS (FlyPort)**	**Raspberry Pi**
Microcontroladores Atmel	Microcontroladores PIC	Microprocesadores ARM (System On-A-Chip)
19,90€ (Arduino Leonardo)	39,00€ (FlyPort Ethernet)	25€ (Raspberry Pi modelo A)
Programación en C/C++ propio o ensamblador en Arduino IDE.	Programación en C en OpenPICUS IDE. Soporte HTML y jQuery en servidor web.	Programación en varios lenguajes según el sistema operativo. Soporte OpenGL ES 2.0.
No soporta sistema operativo. Carga del programa en *flash*.	Sistema operativo FreeRTOS.	Varios sistemas operativos, oficialmente distribuciones de GNU/Linux para ARM.
16 MHz, 32 KB *flash*, 2 KB de *SRAM*, 1 KB de *EEPROM* (Arduino Uno).	PIC de 16 bits, 8 MHz y 16 MIPS, con 256 KB de *flash*.	Arquitectura Von Neumann en SoC: CPU ARM11 a 700 MHz. 256 MiB de SDRAM compartida entre CPU y GPU. GPU VideoCore IV a 250 MHz.
Conexiones USB, I2C, SPI e ICSP.	Conexiones WiFi o Ethernet, USB, USB *On-The-Go*, UART, I2C y SPI.	Conexiones USB host, Ethernet, MicroUSB, GPIO, RCA, HDMI, Jack 3.5mm y RJ45 (sólo modelo B).
VENTAJAS		
Capacidades de control y computación básica.	Capacidades de control, computación y comunicación directa mediante protocolos y servidor web empotrado.	Capacidades de computación de propósito general: operaciones, video, audio, internet, etc.
Muy bajo consumo. Sencillez en la programación. Orientado a usuarios básicos. Comunidad multitudinaria. Extensa documentación y tutoriales. Gran capacidad de extensión mediante *shields*. Arquitectura fácil de construir	Bajo consumo. Sencillez en la programación. Orientado a soluciones comerciales reales y al internet de las cosas. Comunicación integrada. Servidor web empotrado y soporte a protocolos de internet.	Bajo consumo si tenemos en cuenta que es un microrordenador. Varios sistemas operativos completos de PC, con la ventaja que esto supone. Audio/video. Arquitectura ARM muy utilizada y con bastantes

con componentes económicos.	Arquitectura conocida y muy utilizada en multitud de ámbitos. Extensa documentación.	desarrolladores y soporte. Orientado a aprendizaje. Relación precio/prestaciones superior a otras alternativas.
DESVENTAJAS		
Escasa potencia: sólo tareas sencillas y control de sensores/actuadores. Escasa memoria: programas limitados. Poca orientación comercial actual para usos reales. Coste alto en comparación con prestaciones de otras alternativas mencionadas. Necesidad de *shields* para obtener comunicación internet, *bluetooth*, audio limitado, video limitado, controles adicionales, etc. Coste elevado de cada *shield*.	Potencia limitada. Gasto de energía superior en el modelo WiFi. Poco usado en investigación y desarrollo, menor comunidad de soporte. Aunque PIC sí posee una gran comunidad. Coste excesivo en relación precio/prestaciones con otras alternativas. Necesidad de extensiones Nest para obtener más funcionalidad. Coste elevado de cada extensión Nest.	Demasiada potencia y gasto de energía para determinados casos de control sencillo, donde Arduino sería más indicado. Arquitectura ARM11 obsoleta y con menor soporte por parte de terceras empresas. No incorpora WiFi, necesitaría un adaptador WiFi USB compatible con el sistema operativo instalado. No posee almacenamiento masivo integrado, es necesaria una tarjeta SD para el sistema operativo.

Tabla 2: Comparativa entre soluciones de open hardware

Por motivos tecnológicos del proyecto de Telefónica, la arquitectura elegida es Arduino. Para la realización de este proyecto fin de carrera fuera de empresa, el sistema elegido también será Arduino, por las razones aquí expuestas:

- **Implementación realizada en Telefónica**: es la base del proyecto, ya que se trata de un problema real al que se le aporta una solución, a través de los recursos que proporciona Telefónica.

- **Sencillez de programación y diseño** de dispositivos: orientados tanto al uso *amateur* como de investigación, los dispositivos siguen la filosofía *open hardware* y software libre, lo que garantiza y facilita el análisis de su funcionamiento y diseño, así como su programación eficiente.

- Posibilidad de **creación de librerías**: un punto vital para la monitorización de pines de los dispositivos, que dota a los test de una mayor importancia y utilidad.

- Buen número de **pines de salida**, lo que posibilita la **monitorización** de ellos como prueba.

- Proceso de **compilación y carga** mediante línea de comandos con las **herramientas AVR**: unas herramientas estándar y de libre acceso.

- **Multitudinaria comunidad** con un gran soporte y tutoriales, gracias a la filosofía abierta adoptada por Arduino y las plataformas empleadas.

2.3 Sistemas de integración continua

La integración continua, más conocida por su término en inglés *continuous integration*, es una propuesta, ideada por Martin Fowler en un principio, para establecer un modelo de integraciones automáticas que detecten errores en el menor plazo de tiempo posible, originando pruebas de dichas integraciones que se ejecutan en periodos muy cortos de tiempo, habitualmente horas. En otras palabras, se puede definir como la preparación de pruebas automáticas o test automáticos que se ejecutan cada vez que se incorpora una novedad en el proyecto (hablando en términos de código fuente) o cíclicamente. Como se puede ver, la conjunción de un sistema de integración automático con metodologías ágiles o de programación extrema es ideal.

La forma más habitual de implementar el concepto de la integración automática, es trabajar con un repositorio donde los desarrolladores depositen su trabajo, es decir, un sistema de control de versiones. A continuación, se utiliza una aplicación para la planificación de los test, la descarga y compilación del código fuente y la monitorización de su ejecución, estas aplicaciones son el núcleo del sistema y están diseñadas siguiendo las directrices de la integración continua. Ejemplos de estas aplicaciones son Bamboo, Continuum, Hudson, Jenkins, CruiseControl o Team Foundation Build (este último sólo para Microsoft .NET).

Según la plataforma de desarrollo que se esté empleando, existe la posibilidad de utilizar herramientas que facilitan la compilación y ejecución de un proyecto, tales como Ant o Maven para Java, Nant o MSBUILD para Microsoft .NET, *Makefiles* para C/C++ en GNU/Linux, etc., aunque, existen otras herramientas que realizan pruebas unitarias como JUnit para Java o PyUnit para Python que pueden funcionar como complementos ideales para los sistemas de integración. Aparte de esto, y según el sistema de integración escogido, pueden existir extensiones, *plugins* y complementos que aumenten la compatibilidad, funcionalidad y opciones del sistema[38][39].

2.3.1 Método de trabajo

El método de trabajo podríamos resumirlo en los siguientes puntos:

- **Un desarrollador termina de implementar una nueva característica:** esta característica debe ser atómica y sencilla, ya que un concepto de la integración continua es realizar *testing* sobre código mínimo que realice operaciones sencillas para asegurar su estabilidad y que bajo ningún supuesto existirá un error en ese punto. En otras palabras, si funciones sencillas no fallan, los cimientos del resto serán lo suficientemente estables y nos ahorrará tiempo de búsqueda de errores, que perderíamos al implementar todo sin realizar pruebas intermedias.

- **La característica terminada se añade o actualiza en el sistema de control de versiones:** en este punto, el sistema de integración puede detectar el cambio o bien estar configurado para ejecuciones cíclicas en un intervalo de tiempo. En cualquier caso, el sistema de integración descarga una copia de la última versión del proyecto.

- **Comienza la prueba o testeo:** el sistema de integración compilará la última versión y la ejecutará mediante test que estimulen la aparición de errores, mientras tanto, monitorizará todo e informará de errores de compilación, ejecución o resultados no esperados.

- **Informe de la prueba:** si el sistema determina un resultado incorrecto, se notificará según su configuración (existen multitud de opciones: aviso sonoro, *e-mail*, *tweet*, etc.), en caso contrario, notificará que el nuevo código ha sido integrado con éxito y es estable.

2.3.2 Ventajas

Existen varias ventajas en el uso de este modelo de desarrollo:

- Los desarrolladores pueden detectar y **solucionar problemas de integración de forma continua**, evitando largas pérdidas de tiempo de depuración cuando se acercan las fechas de entrega.

- **Disponibilidad constante** de una compilación para pruebas, demos o lanzamientos anticipados.

- **Ejecución inmediata** de las pruebas unitarias de forma automática.

- **Monitorización continua de las métricas** de calidad del proyecto.

La integración continua proporciona automatización y estabilidad, ahorra tiempo y dinero en realizar pruebas al final y rebuscar dónde se encuentran los errores y proporciona un sistema de control del código fuente. Por sus ventajas, no es extraño que empresas importantes apuesten por este modelo como Microsoft, IBM u Oracle.

2.3.3 Desventajas

Evidentemente, existen claras desventajas[40][41]:

- **Los problemas de integración se deben detectar cuando son insertados en el código,** pero esto puede originar la integración en conjunto del proyecto no sea la más optimizada. Por lo que es necesario realizar siempre pruebas totales al final de su desarrollo.

- Los desarrolladores **no poseen el hábito de actualizar** el repositorio con frecuencia.

- **La compilación de código se puede demorar** y tomarse un impedimento para la construcción e integración continua del código fuente.

- Estos sistemas **benefician el correcto funcionamiento, no la optimización,** clave en determinados campos.

- La **instalación**, **mantenimiento** de estos sistemas y la **realización de pruebas** conlleva una buena parte del **tiempo** de los desarrolladores, o mantener una **persona dedicada**.

2.3.4 Alternativas y comparativa

En este apartado, se analizan las alternativas más conocidas en entornos de integración continua.

2.3.4.1 CruiseControl

Ilustración 15: Logotipo de CruiseControl

CruiseControl se presenta como un sistema de integración continua, y como un *framework* para crear procesos de compilación continuos personalizados. Está desarrollado en Java, es gratuito y de código abierto bajo licencia de tipo BSD. Provee una interfaz web para la administración, gestión y monitorización. Posee un sistema de extensiones para soportar una buena variedad de plataformas, herramientas de compilación, esquemas de notificación, etc., aunque inicialmente soporta Ant, NAnt, Maven, Phing, Rake y Xcode, además de línea de comandos de consola del sistema operativo y *scripts*. Existen versiones para Ruby y plataforma .NET (CruiseControl .NET).

Su desarrollo original partió de ThoughtWorks, surgiendo como una necesidad de una solución de integración continua para sus proyectos. Posteriormente, debido a su utilidad, se amplió su desarrollo como una aplicación independiente[42].

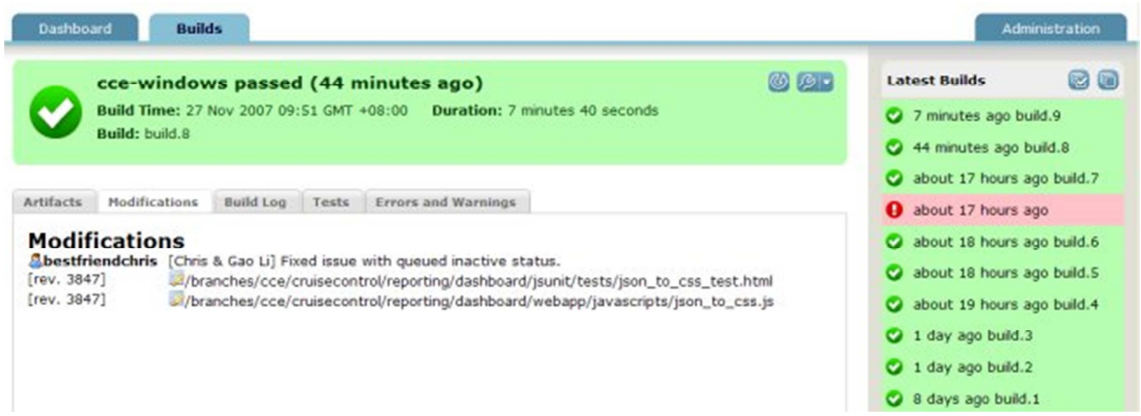


Ilustración 16: Dashboard de CruiseControl

En su diseño, consta de tres módulos principales:

- **Ciclo de compilación:** es el corazón del sistema, y se encarga, mediante disparadores, de compilar la versión correspondiente y realizar las notificaciones a los usuarios pertinentes mediante diversas técnicas de publicación. Los disparadores pueden ser internos, programados en un intervalo o cuando existan cambios en el sistema de control de versiones, o externos. La configuración se estructura en un fichero XML.

- **Aplicación JSP:** proporciona una interfaz web que permite a los usuarios navegar por los resultados de los ciclos de compilación y las versiones compiladas.

- ***Dashboard*:** proporciona una representación visual del estado de los proyectos.

Es posible controlar el sistema y monitorizar los proyectos desde una aplicación externa mediante HTTP y Java RMI[43].

2.3.4.2 Jenkins

Ilustración 17: Logotipo de Jenkins

Jenkins es un sistema de integración continua desarrollado en Java bajo licencia MIT y *Creative Commons*. Es gratuito y de código abierto, además de multiplataforma. Está basado en el proyecto Hudson, ya que es un clon o, simplemente, un cambio de nombre, ocasionado por una disputa con Oracle, ya que ésta reclama los derechos de uso de la marca Hudson. Jenkins está construido como un *servlet* para Apache Tomcat, aunque puede instalarse y funcionar como un servicio del sistema independiente. Soporta CVS, Subversion, Git, Mercurial, Perforce y Clearcase, sin contar extensiones, y puede ejecutar Ant y Maven, aparte de *scripts* y ficheros de procesamiento por lotes de Windows o comandos de consola. Fue creado por Kohsuke Kawaguchi como una alternativa a CruiseControl y otros sistemas existentes, recibiendo en 2011 un premio Google-O'Reilly Open Source Award.

Posee una gran variedad de extensiones y complementos que le hacen soportar casi cualquier opción de desarrollo disponible, variar maneras de notificar errores, lograr la integración con bases de datos, personalizar la interfaz, y hasta colocar juegos en el sistema y demás curiosidades. Por lo tanto, la compatibilidad de Jenkins con cualquier plataforma de desarrollo está asegurada, ya que es posible crear extensiones a medida o realizar *scripts* personalizados para su ejecución[44].

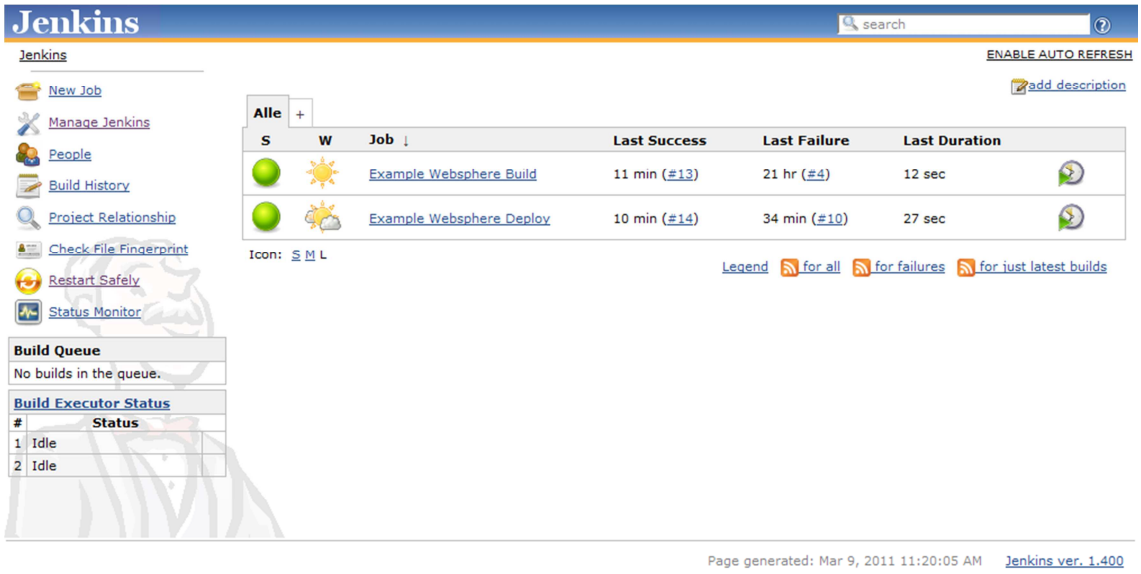


Ilustración 18: Interfaz web principal de Jenkins

La interfaz gráfica de Jenkins consiste en un sitio web accesible con cualquier navegador. Permite administrar todo el sistema (extensiones, configuración, repositorios, opciones de sistema operativo, seguridad, etc.), crear y gestionar proyectos y ciclos de compilación, monitorizar ejecuciones, consultar historiales, etc., las opciones son muy variadas y provocan que el sistema sea totalmente flexible. Posee aplicación móvil para iOS y Android[45].

2.3.4.3 Bamboo

Ilustración 19: Logotipo de Bamboo

Bamboo es un sistema de integración continua desarrollado por Atlassian. Se distribuye bajo licencia propietaria, es multiplataforma y de pago, aunque dispone de versión gratuita para actividades no comerciales y de código abierto y posee descuentos para instituciones académicas. Bamboo soporta compilaciones en cualquier lenguaje de programación utilizando cualquier herramienta de compilación tipo xUnit o Selenium, incluyendo soporte de Ant, Maven, *makefiles*, Microsoft .NET y comandos de consola del sistema operativo, al estilo de Jenkins. Permite integración con *IDEs* basados en Eclipse o IntelliJ IDEA, además de Visual Studio[46].

Cada *check-in* realizado por un programador es testeado continuamente, como en cualquier sistema del estilo, y es compatible con Git, Mercurial, Perforce, CVS y Subversion, incluye la novedosa característica de planificación automática de ramas y reintegración continua de ellas. Posee múltiples configuraciones y herramientas para las notificaciones: e-mail, mensaje instantáneo, RSS o incluso en el propio IDE, entre otras. Entre sus características más avanzadas, permite ejecutar ciclos de compilación en Amazon EC2 Cloud, en paralelo con otros equipos, así como paralelamente en un mismo equipo mediante tuberías. Soporta integración con JIRA, un gestor de proyectos desarrollador por la misma compañía, notificando los resultados de los test. Además, incluye su propio gestor de versiones que agiliza la entrega de versiones a cliente y su actualización. Se integra totalmente con herramientas de control de calidad del código fuente como Sonar y Clover, aparte de notificar qué cambio del código ha ocasionado el error. Posee un apartado específico de métricas y estadísticas del proyecto.

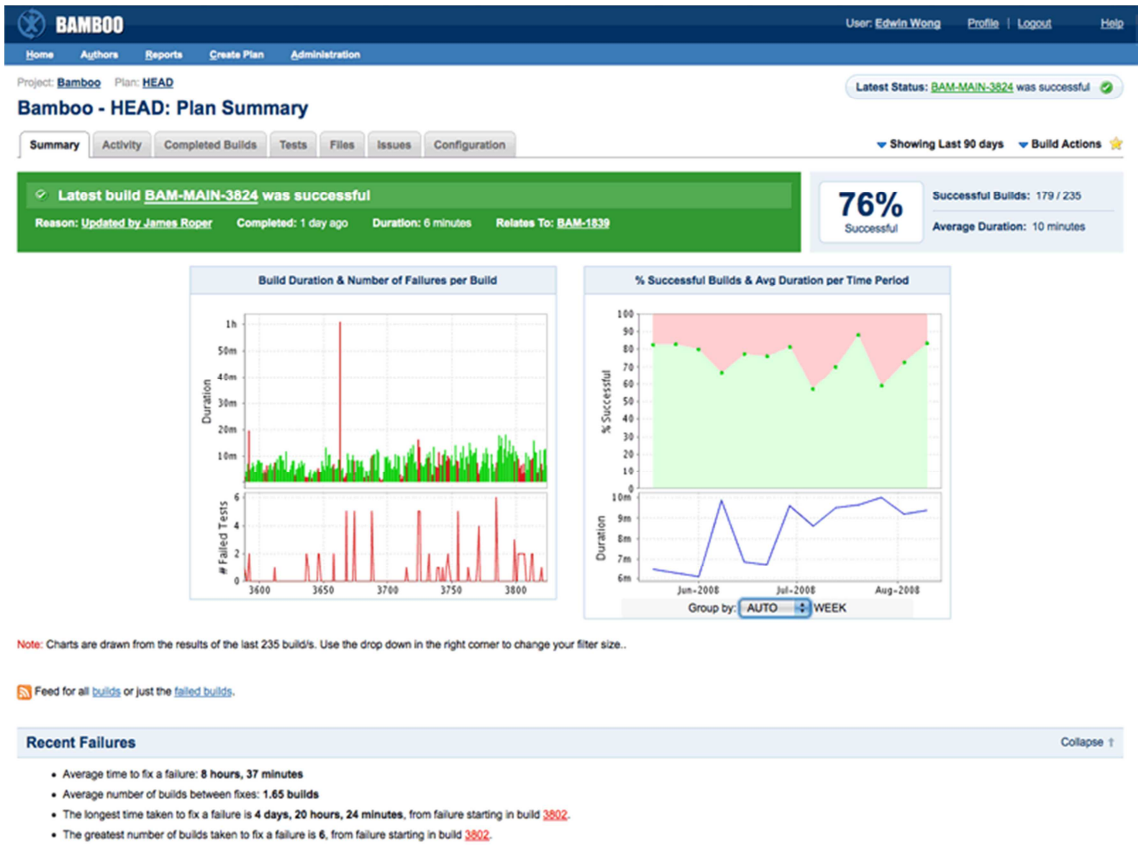


Ilustración 20: Interfaz web de resumen y métricas de la versión HEAD en Bamboo

La interfaz se basa en un servidor propio con su propio sitio web de gestión, donde se puede controlar cualquier aspecto del sistema de integración, así como sus compilaciones, testeos y proyectos.

En cuanto a las extensiones, posee conectores para servicios de alojamiento de código fuente como GitHub o Bitbucket, además de un sistema de integración con Microsoft Sharepoint, que se puede obtener aparte. Por otra parte, existen extensiones, por nombrar algunas de las más conocidas, para automatizar ficheros WAR y EAR (LiveRebel), integrar el sistema como parte del proceso de desarrollo (DeployIt), integrar almacenamiento con Amazon S3 y compilación con Amazon EC2, permitir el desarrollo en la nube con Git (Heroku), mejorar la eficiencia mediante la nube en los *tests* de Selenium (SauceLabs) o detener compilaciones erróneas para obtener trazas (HungBuildKiller).

Atlassian provee una herramienta para el desarrollo de extensiones, denominada Atlassian SDK. La comunicación con el servidor de Bamboo se realiza utilizando un API REST[47].

2.3.4.4 Continuum

Ilustración 21: Logotipo de Continuum

Continuum es un sistema de integración continua, basado en un servidor y gestionado desde un servidor web, al estilo de sus competidores Jenkins o Bamboo. Está desarrollado por Apache bajo lenguaje Java con la plataforma de gestión Maven, siendo totalmente integrado con ella, también de Apache. Se distribuye bajo licencia Apache 2.0 y es de código abierto y multiplataforma. Al igual que sus competidores más directos, es capaz de realizar compilaciones planificadas automáticamente e informar de errores mediante correo electrónico. Al igual que Jenkins o Hudson, puede ejecutarse como una aplicación independiente o funcionar como un WAR introducido en un soporte de *servlets*.

Aparte de la interfaz web de gestión, su configuración de proyectos se basa en un fichero XML, a través del cual se pueden añadir los proyectos deseados, y el sistema se encargará del resto, desde comprobar el repositorio de versiones a ejecutar las pruebas unitarias. Soporta CVS, Subversion, Clearcase, Perforce, Starteam, Microsoft Visual SourceSafe, CM Synergy, Bazaar y Mercurial, además de almacenar y mostrar los cambios en los sistemas de control de versiones para compilación. Es posible notificar resultados por *e-mail*, Jabber, Google Talk, Windows Live Messenger e IRC, entre otros. Las ejecuciones pueden realizarse mediante las herramientas Maven, Ant o a través de *scripts* de consola. Como características avanzadas, incorpora soporte para compilaciones paralelas y/o distribuidas.

No permite la extensión de funcionalidades mediante *plugins*, sin duda, un inconveniente que le perjudica a la hora de elegir, es un producto relativamente joven y que necesita tiempo de mejora. Sin embargo, posee una API XMLRPC para la interactuación con aplicaciones externas[48][49].

2.3.4.5 Comparativa

A continuación, se expone una comparativa de las alternativas estudiadas según su licencia y precio, compatibilidad con herramientas de compilación, compatibilidad con sistemas de control de versiones, procesos de comunicación externa, posibilidades de ampliación y soporte de procesamiento distribuido.

Características comunes como la compatibilidad con herramientas de pruebas unitarias, o características expandibles, no se incluyen en la comparativa.

cruisecontrol	Jenkins	Bamboo	continuum
Gratuito y de código abierto	Gratuito y de código abierto	De pago y propietario (desde $10 hasta $16000 según necesidades)	Gratuito y de código abierto
Ant, NAnt, Maven, Phing, Rake, Xcode, .NET, Ruby, *scripts* y *shell*.	Ant, Maven, *scripts* y *shell*.	xUnit, Selenium, Ant, Maven, *makefiles*, Microsoft .NET, Eclipse, IntelliJ IDEA, *scripts* y *shell*.	Ant, Maven, *scripts* y *shell*.
Accurev, Clearcase, CMSynergy, jCVS, MKS, Perforce, Pvcs, StarTeam, Subversion, View CVS y CVS.	CVS, Subversion, Git, Mercurial, Perforce y Clearcase.	Git, Mercurial, Perforce, CVS y Subversion.	CVS, Subversion, Clearcase, Perforce, Starteam, Microsoft Visual SourceSafe, CM Synergy, Bazaar y Mercurial.
Incluye *framework*. Comunicación externa bajo HTTP y Java RMI.	Comunicación externa bajo API XML y aplicaciones móviles Android & iOS.	Comunicación externa bajo API REST.	Comunicación externa bajo API XMLRPC.
Posibilidades de extensión con *plugins*. Desarrollo bajo Java.	Posibilidades de extensión con *plugins*. Desarrollo bajo Java.	Posibilidades de extensión con *plugins*. Desarrollo con Atlassian SDK.	No permite extensiones.
Gestión de ejecuciones distribuidas del sistema.	Compilaciones y testeo distribuidos.	Distribución total, compatibilidad con Amazon EC2 para cómputo y Amazon S3 para almacenamiento.	Compilaciones distribuidas.

Tabla 3: Comparativa entre sistemas de integración continua

A razón del tutor del proyecto en Telefónica, y por necesidades de la empresa, el sistema elegido es Jenkins. Para la realización de este proyecto fin de carrera fuera de empresa, el sistema elegido también será Jenkins, por las razones aquí expuestas:

- **Gratuito, fácil instalación y autoconfiguración.**
- Gran **comunidad de soporte y extendido uso** del sistema.
- **Compatibilidad con Hudson.**
- **Compatibilidad con Subversion y *scripts* de Python.**
- Posibilidad de instalarse como un **servicio de Windows**, o administración mediante **Apache Tomcat**.
- Numerosas **extensiones de funcionalidad** de múltiples ámbitos (compilación, notificación, métricas, aplicaciones móviles, etc.).

2.4 Entorno de la empresa

Este proyecto fin de carrera se ha implementado en el **laboratorio *Physical Internet Lab***, perteneciente al área de emergentes de PDI (*Product Development & Innovation*), anteriormente conocido como Telefónica I+D, dentro de Telefónica Digital. El objetivo del laboratorio es democratizar el internet de las cosas, es decir, proporcionar soluciones M2M que cualquier persona, desde un particular hasta una gran empresa, pueda utilizar sin complicaciones y de forma efectiva, aprovechando la extensa red de comunicaciones del grupo Telefónica.

En este último aspecto, Telefónica ha diseñado un *shield* o escudo para Arduino que incorpora un módem GSM/GPRS y un portador de tarjeta SIM, con el objetivo de proporcionar una comunicación inalámbrica real a cualquier Arduino mediante la red 2G, incluyendo servicios de comunicación tradicionales como llamadas o SMS, o servicios de conexión TCP mediante sockets. Para ello, se ha diseñado e implementado una librería de utilización que facilita el uso de los recursos del *shield* mediante los servicios mencionados. De esta manera, mediante una simples funciones, y cumpliendo con la premisa de facilidad y cercanía a todo el mundo que propone la filosofía Arduino, se pueden programar placas Arduino que, por ejemplo, se conecten a un servidor tipo Cosm o Pachube, a un servidor web, etc.

El interés de proporcionar este *shield* viene dado por el motivo de que Arduino es la plataforma de open hardware cuya utilización ha crecido exponencialmente en los últimos años, siendo adquirida por investigadores, por empresas, por universidades, por amantes del hardware y la electrónica y por simples usuarios básicos que ven muchas posibilidades abiertas en la filosofía Arduino y sus prestaciones. Su gran fama y comunidad de aficionados, además de su utilización por varias empresas como hardware base para diversos productos, hace que proporcionar una comunicación móvil real sea muy interesante para Telefónica.

Por otro lado, en Telefónica Digital, existe una fuerte creencia en las metodologías ágiles, sobre todo en SCRUM y las basadas en metodología Toyota o JIT. Es por ello, que la integración continua se presenta como una necesidad, debido a la continua evolución de la librería GSM y sus ejemplos, con cada vez más adaptaciones a diversos propósitos y mejoras de administración del modem. Entonces, se podría realizar una integración continua de la librería mediante un sistema compatible con el desarrollo en C/C++ que testeara todas sus posibilidades, pero esto sería ineficiente y poco útil, ya que la complejidad de sus estructuras de datos, utilizando buffers circulares, y la fuerte dependencia del hardware del modem complicarían los test hasta límites intolerables donde dependeríamos de simulaciones del modem, algo que excede cualquier cota de tiempo y dedicación de cualquier proyecto. No olvidemos que el objetivo de la librería es la abstracción de los servicios proporcionados por el modem GSM/GPRS, ocultando comandos AT bajo funciones simples y gestionando respuestas del modem proporcionando al *sketch* principal la información que necesita.

En este punto, la mejor forma de testear con ejemplos el correcto funcionamiento de la librería es ejecutar pruebas con el hardware presente, como haría cualquier persona en su casa. Sin embargo, nos encontramos con el problema del tiempo y la dedicación de una persona de forma especial para este cometido, y por lo tanto, se echa en falta la existencia de un sistema de integración continua que funcione testeando código directamente sobre el hardware.

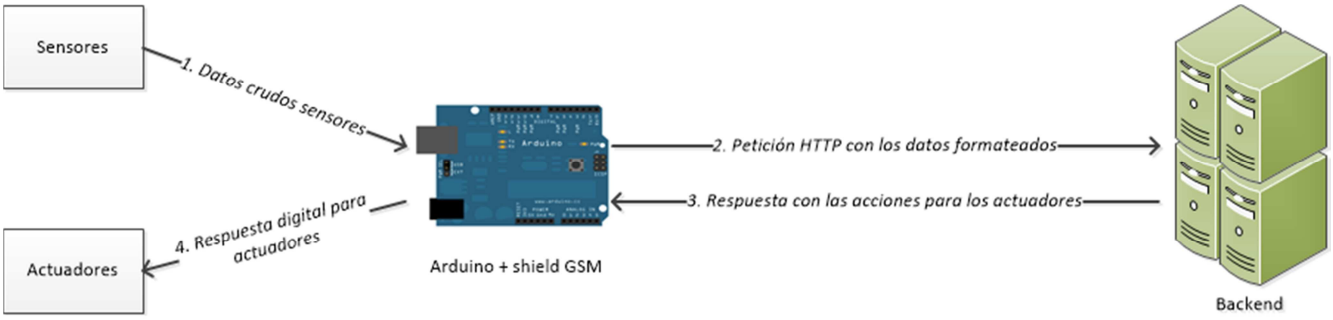


Ilustración 22: Esquema del proceso de computación con backend

Es interesante la idea de que la inteligencia está en el *backend*, muy utilizada en la arquitectura de Arduino como simple capturador de sensores y actuador. En este concepto, Arduino posee todos los sensores necesarios, y todos los actuadores deseados para cada evento destinado a producirse. Arduino se conecta mediante internet a un *backend* y le deja información sobre los sensores mediante una petición HTTP, entonces, el *backend* contiene la lógica y la computación compleja, y el usuario puede manejarlo con un *frontend*. De esta manera, una vez procesada la información, el *backend* responde a Arduino con la información correspondiente para sus actuadores (relés, diodos led, etc.). Para *backend*, en Telefónica se apuesta fuertemente por la plataforma Django, que proporciona un servidor web muy pensado para este tipo de tareas. El lenguaje de programación utilizado es para el desarrollo en Django suele ser Python. Por este motivo, el código necesario para la integración continua será programado en Python, facilitando así el soporte y la posibilidad de reutilización e integración, además de ser ideal para el manejo de cadenas de caracteres en cuanto a reconocimiento y comparación, así como para el manejo de sistemas de ficheros.

Existiendo sistemas de integración continua para el software, que son compatibles, la mayoría, con *scripts* y comandos del sistema operativo siendo posible planificarlos en el tiempo, y dada la característica multiplataforma de Python, aparte del soporte a los sistemas de control de versiones más conocidos, se decidió crear un *script* o programa para realizar los test y los comunicara al sistema de integración continua existente, pudiendo integrar software de distintas plataformas y hardware en un mismo sistema. Por lo tanto, este proyecto es causante de la instalación de un sistema de integración continua en el laboratorio.

El sistema de control de versiones utilizado en Telefónica Digital es Subversion, tal y como se explica en el apartado 2.3.4.5 por las ventajas mencionadas.

En resumen, la tecnología a utilizar, relevante para esta solución, es la siguiente:

- Plataforma de open hardware **Arduino**.
- Lenguaje de programación **Python**.
- Sistema de integración continua **Jenkins**.
- Sistema de control de versiones **Subversion**.

3. Análisis de la solución

A continuación, se describe y analiza la solución planteada y se estudian los requisitos exigidos por el cliente, en este caso, la empresa, así como de sus casos de usos derivados, imprescindibles para garantizar un funcionamiento deseado y adaptado a las necesidades propuestas.

3.1 Descripción de la solución

En este apartado, se aborda la descripción del proyecto, especificando sus características elementales, su objetivo, a quién va destinado y su contexto.

3.1.1 Capacidades generales

En esta sección, se especifican las capacidades generales de la solución de este proyecto de una manera global.

El objetivo de la solución propuesta es integrar test sobre open hardware, en este caso Arduino, de manera que se automaticen pruebas sobre ciertos aspectos de una librería, sobre un *sketch* o sobre el propio hardware, arrojando los resultados pertinentes, y siendo totalmente integrable en los sistemas de integración continua dedicados al software actuales.

Para lograr el objetivo, la solución debe ser capaz de actualizar las librerías y *sketches* de prueba a la última versión disponible, compilar los *sketches* de prueba y cargarlos en las placas hardware Arduino, y monitorizar su comportamiento. Además, se ofrece la opción de utilizar un Arduino para monitorizar la E/S básica de otro, mediante una modalidad especial que denominaremos Testduo, siempre monitorizando los sucesos e informando de errores y resultados inesperados.

Como se puede intuir, existen diferentes fases en un ciclo de prueba. Cada componente de la solución cumple con una función:

- **Sistema de control de versiones (Subversion):** integra una estructura organizativa de librerías y *sketches* de prueba. Su misión es la misma que la de cualquier sistema de control de versiones y nos permite mantener repositorio central en red que puede ser utilizado por varias soluciones de integración distribuidas o por diversos desarrolladores.

- **Sistema de integración continua (Jenkins):** gestiona la integración, es decir, la ejecución de ciclos de prueba, cuando se detecten cambios en el repositorio o mediante una planificación fija, y la notificación de los resultados de las pruebas. La utilización de un sistema de integración continua de software actual nos permite integrar las pruebas de Arduino con otras pruebas de un desarrollo puramente software en un mismo sistema, así como utilizar los numerosos *plugins* y extensiones disponibles para extender su funcionalidad de manera casi ilimitada.

- **Programa de adaptación a Arduino (Testduino):** gestiona un ciclo de prueba en su parte técnica, en otras palabras, se encarga de la compilación de los *sketches* de prueba, su carga en las placas Arduino, y monitorización de sus resultados, notificando el resultado final al sistema de integración continua. En mayor detalle, debe ocuparse de aspectos como la gestión de los puertos COM de comunicación con las placas Arduino, las librerías necesarias para cada *sketch*, así como de la compilación y carga del *sketch* de monitorización de E/S en la modalidad Testduo, donde se monitoriza un Arduino con ayuda de otro conectado a su E/S.

Por supuesto, cada componente puede requerir de ciertas plataformas para su ejecución, que se detallan posteriormente.

Desde el punto de las fases a realizar en cada prueba, podemos distinguir los siguientes subprocesos principales:

- **Subproceso de planificación:** gestionado por Jenkins, debe comprobar la configuración del proyecto en Jenkins, encargándose de actualizar la copia local del repositorio cuando existan diferencias, manteniendo así la última versión para los test. Por otra parte, debe ejecutar el programa Testduino cuando esté planificado.

- **Subproceso de compilación y carga:** gestionado completamente por Testduino, se encarga de administrar las comunicaciones con las placas hardware y los *sketches* de prueba, compilándolos con sus librerías y cargándolos en las placas correspondientes cuando estén disponibles.

- **Subproceso de monitorización:** monitoriza los resultados y trazas de las pruebas por los puertos series correspondientes a las placas en funcionamiento. Al finalizar, envía los resultados finales a Jenkins, determinando si la prueba total ha sido un éxito o a fracasado, y por lo tanto, notificando Jenkins este resultado según su configuración.

3.1.2 Restricciones generales

A continuación, se describen las restricciones globales de la solución, que ayudan a garantizar su correcto funcionamiento y su adaptación a las necesidades exigidas:

- La solución debe proporcionar una **interfaz de comunicación** con las placas hardware **Arduino**, esta interfaz es **serie**, a través de los puertos COM del equipo. Esta configuración es impuesta por las especificaciones de Arduino. La comunicación debe ser **bidireccional** y debe garantizar la correcta interpretación de los resultados de las pruebas.

- **Testduino** dispondrá de **dos modalidades** de prueba exclusivas entre sí: **Testduino Single**, que es la modalidad tradicional donde se **monitoriza el resultado que un solo Arduino arroja** por la comunicación serie al ejecutar un *sketch*, y **Testduo**, que proporciona una **monitorización de los resultados de E/S** que arroja por los pines E/S la ejecución de un *sketch*, utilizando un **segundo Arduino como monitor** conectando sus E/S.

- Es **obligatoria la utilización de las herramientas del IDE de Arduino**, por lo que debe encontrarse **instalado**, para así asegurar el seguimiento del procedimiento oficial de carga de programas en placas Arduino.

- **Testduino** debe ser **ejecutable por consola de MS-DOS o Windows PowerShell**, asegurando su compatibilidad con la plataforma Microsoft Windows, así como su **integración** mediante comando en cualquier otro sistema de integración continua al estilo Jenkins.

3.1.3 Características del usuario

El perfil de usuario objetivo de la solución será un técnico o ingeniero, con conocimiento medio-avanzados informáticos generales.

Para la utilización frecuente, solo sería necesario navegar por un sitio web, saber utilizar un repositorio de versiones a nivel básico y editar una configuración básica de Testduino sobre Arduino. Aunque, siempre es recomendable que, en caso de problemas, el usuario disponga de un conocimiento más avanzado:

- Conocimiento básico-medio sobre **Python**.
- Conocimiento básico sobre **Apache Tomcat**.
- Conocimiento medio-avanzado sobre la plataforma **Arduino**.
- Conocimiento avanzado sobre sistemas **Microsoft Windows**.
- Conocimiento medio sobre sistemas **Subversion**.

3.1.4 Entorno operacional

En este apartado, se analizan las tecnologías del sistema en torno a los usuarios definidos en el apartado anterior.

Para cualquier tipo de usuario, el entorno operacional del programa de adaptación de Testduino y Jenkins será el siguiente:

Ordenador PC/Mac con sistema operativo Microsoft Windows 7 o superior (32 o 64 bits) y conexión constante a internet de alta velocidad (>256 Kbps de descarga). *Será necesaria la instalación de un explorador de internet (preferiblemente Internet Explorer, Mozilla Firefox, Google Chrome, Opera o Safari) y Apache Tomcat si se desea ejecutar Jenkins como un servlet tradicional.* *Para Testduino, será necesario disponer de Python 3 y las librerías PySerial y PyWin32 instaladas en 32 bits, así como una copia del IDE oficial de Arduino en su versión 1.0.1 o superior.*

Tabla 4: Entorno operacional de cualquier usuario

El servidor de Subversion dispondrá de su propio equipo, según las especificaciones mínimas proporcionadas en su documentación.

3.2 Análisis de requisitos

A continuación, se detallan los requisitos obtenidos, tanto de usuario como de software, así como los casos de uso relacionados. En el análisis de requisitos, la referencia a sistema quiere decir a la solución conjunta completa.

3.2.1 Requisitos de usuario

En esta parte, se especifican los requisitos de usuario, necesarios para poder diseñar la solución adecuada a las necesidades del usuario.

Estos requisitos se dividen dos tipos: capacidad, que indican qué debe ser capaz de realizar la solución, y de restricción, que señalan las restricciones que establecen el cómo se deben realizar los objetivos establecidos por los requisitos de capacidad.

Los requisitos de restricción se dividen en subtipos:

- **Requisitos de interfaz de comunicación:** especifican la estructura comunicativa entre componentes de la solución y con el hardware.
- **Requisitos de interfaz hardware:** especifican el tipo de relación entre las placas hardware y su manejo por parte del software.
- **Requisitos de interfaz software:** aportan características de funcionamiento de los diferentes componentes de la solución y la plataforma que la sostiene.
- **Requisitos de interfaz de usuario:** especifican elementos necesarios para la relación de la solución con el usuario.
- **Requisitos de portabilidad:** detallan características que permitan la portabilidad de la solución.
- **Requisitos de seguridad:** especifican procedimientos restrictivos relacionados con la seguridad de la solución.
- **Requisitos de tiempo:** abogan restricciones temporales que se deben aplicar a las capacidades.

El formato de plantilla utilizado en la especificación es el siguiente:

Identificador:	XX-UR-YY
Descripción:	
Necesidad:	
Prioridad:	
Estabilidad:	
Origen:	
Grado de verificabilidad:	

Tabla 5: Plantilla de requisito de usuario

- **Identificador:** es el código único que identifica a cada requisito. Se compone de 3 partes:
    - **XX:** es CA si el requisito es de capacidad o RE si es de restricción.
    - **UR:** responde a la sigla *User Requirements*.
    - **YY:** número de requisito.
- **Descripción:** detalla la especificación del requisito de manera precisa y simple.
- **Necesidad:** establece la importancia del requisito desde el punto de vista del cliente. Los valores pueden ser esencial, conveniente u opcional.
- **Prioridad:** establece la importancia del requisito en función del desarrollo del proyecto. Los valores pueden ser alta, media o baja.
- **Estabilidad:** indica el grado de variación que puede experimentar el requisito a lo largo del desarrollo del proyecto. Los valores pueden ser no cambia, cambiante o muy inestable.
- **Origen:** indica la procedencia del requisito. El valor cliente refleja que el requisito proviene del análisis de los requerimientos exigidos por el cliente, por el contrario, el valor proyecto establece que el requisito ha sido incorporado por idea del propio desarrollador del proyecto.
- **Grado de verificabilidad:** mide la dificultad que posee la verificación del cumplimiento del requisito. Los valores pueden ser alta, media o baja.

3.2.1.1 Requisitos de capacidad

Identificador:	CA-UR-01
Descripción:	El sistema debe ser capaz de mantener sus librerías a probar y sus *sketches* de prueba siempre actualizados.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 6: CA-UR-01

Identificador:	CA-UR-02
Descripción:	El sistema debe ser capaz de ejecutar las pruebas unitarias bajo temporizador/programación.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 7: CA-UR-02

Identificador:	CA-UR-03
Descripción:	El sistema debe disponer la posibilidad de ejecutar las pruebas unitarias al detectar cambios de versión en el repositorio.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 8: CA-UR-03

Identificador:	CA-UR-04
Descripción:	El usuario tiene que ser notificado de la ejecución y resultado de las pruebas.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 9: CA-UR-04

Identificador:	CA-UR-05
Descripción:	El usuario puede realizar pruebas de librerías con sus *sketches* de prueba o de sólo *sketches* de pruebas.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 10: CA-UR-05

Identificador:	CA-UR-06
Descripción:	El sistema debe poder identificar los *sketches* de las pruebas.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 11: CA-UR-06

Identificador:	CA-UR-07
Descripción:	El usuario tiene que poder insertar trazas como comentarios en los *sketches* estableciendo órdenes sobre las pruebas.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 12: CA-UR-07

Identificador:	CA-UR-08
Descripción:	El usuario puede probar un *sketch* en un puerto específico.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	No cambia
Origen:	Proyecto
Grado de verificabilidad:	Alta

Tabla 13: CA-UR-08

Identificador:	CA-UR-09
Descripción:	El usuario puede establecer una monitorización de pines mientras sucede la ejecución de un *sketch*.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Proyecto
Grado de verificabilidad:	Alta

Tabla 14: CA-UR-09

Identificador:	CA-UR-10
Descripción:	El sistema debe poder omitir *sketches* así especificados por el usuario.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 15: CA-UR-10

Identificador:	CA-UR-11
Descripción:	El sistema debe ser capaz de comunicarse con el microcontrolador de varias placas Arduino simultáneamente.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 16: CA-UR-11

Identificador:	CA-UR-12
Descripción:	El usuario debe poder configurar las rutas de acceso, parámetros comunicación y puertos serie disponibles detalladamente.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 17: CA-UR-12

Identificador:	CA-UR-13
Descripción:	El sistema debe disponer de mecanismos de cronometraje que eviten bloqueos de procesos.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 18: CA-UR-13

Identificador:	CA-UR-14
Descripción:	El sistema debe poder analizar los *sketches* buscando trazas para modificaciones en el modo de ejecución.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 19: CA-UR-14

Identificador:	CA-UR-15
Descripción:	El sistema debe ser capaz de realizar el proceso de compilación y carga realizado por Arduino IDE para las placas UNO y MEGA ADK *for Android*.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 20: CA-UR-15

Identificador:	CA-UR-16
Descripción:	El sistema debe detectar palabras clave y trazas que indiquen información de pines, situación de error o fin con éxito de la ejecución.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 21: CA-UR-16

3.2.1.2 Requisitos de restricción

3.2.1.2.1 Requisitos de interfaz de comunicación

Identificador:	RE-UR-01
Descripción:	Los procesos de monitorización de los *sketches* se ejecutan en paralelo mediante multi-hilo.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 22: RE-UR-01

Identificador:	RE-UR-02
Descripción:	La comunicación con las placas Arduino se realiza a través de comunicación serie por sus puertos correspondientes.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 23: RE-UR-02

Identificador:	RE-UR-03
Descripción:	La comunicación entre el sistema de integración y el programa de adaptación Testduino consiste en la ejecución del propio Testduino y su valor de retorno.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 24: RE-UR-03

Identificador:	RE-UR-04
Descripción:	La solución se sincronizará con el repositorio de versiones mediante conexión TCP/IP.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 25: RE-UR-04

3.2.1.2.2 Requisitos de interfaz hardware

Identificador:	RE-UR-05
Descripción:	La comunicación entre placas hardware Arduino para la monitorización de pines consiste en la conexión de ellos extremo a extremo.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 26: RE-UR-05

Identificador:	RE-UR-06
Descripción:	El microcontrolador debe tener habilitada una salida serial para la comunicación serie a unos baudios por defecto.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 27: RE-UR-06

Identificador:	RE-UR-07
Descripción:	Para un funcionamiento efectivo, debe existir al menos una placa Arduino de prueba con conexión serie estable.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 28: RE-UR-07

3.2.1.2.3 Requisitos de interfaz software

Identificador:	RE-UR-08
Descripción:	La solución debe componerse de: • Testduino (programa de adaptación). • Jenkins (integración continua). • Subversion (repositorio de versiones).
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 29: RE-UR-08

Identificador:	RE-UR-09
Descripción:	La implementación de Testduino debe realizarse en lenguaje Python.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 30: RE-UR-09

Identificador:	RE-UR-10
Descripción:	Testduino almacena los parámetros de configuración en dos ficheros de configuración: testconfig y testports.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 31: RE-UR-10

Identificador:	RE-UR-11
Descripción:	Testduino aprovecha la configuración y se basa en la rutina de compilación y carga realizada por el IDE oficial de Arduino.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 32: RE-UR-11

Identificador:	RE-UR-12
Descripción:	La monitorización entre dos placas Arduino se realiza siguiendo un sistema de trazas que informen sobre los pines y su resultado esperado, y puntos de ruptura en el *sketch* a espera de resultados.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 33: RE-UR-12

3.2.1.2.4 Requisitos de interfaz de usuario

Identificador:	RE-UR-13
Descripción:	La interfaz gráfica de control de la solución es el panel de administración de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 34: RE-UR-13

Identificador:	RE-UR-14
Descripción:	Testduino posee la apariencia de un *script* de consola, interactuando con el usuario/Jenkins a través de mensajes de consola.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 35: RE-UR-14

Identificador:	RE-UR-15
Descripción:	Testduino debe arrojar por consola las pruebas en curso y su comunicación serie recibida.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 36: RE-UR-15

3.2.1.2.5 Requisitos de portabilidad

Identificador:	RE-UR-16
Descripción:	El programa Testduino debe ser ejecutable autónomamente.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 37: RE-UR-16

3.2.1.2.6 Requisitos de seguridad

Identificador:	RE-UR-17
Descripción:	Los procesos internos se realizan en carpetas temporales e internas de trabajo.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 38: RE-UR-17

3.2.1.2.7 Requisitos de tiempo

Identificador:	RE-UR-18
Descripción:	El sistema debe detenerse después de un tiempo crítico máximo de ejecución o de monitorización.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	Cliente
Grado de verificabilidad:	Alta

Tabla 39: RE-UR-18

3.2.2 Casos de uso

Los casos de uso son la representación gráfica de los requisitos. Su objetivo es representar y detallar las funcionalidades y tareas que puede realizar un usuario con la solución, partiendo de la base de que el usuario controla el sistema mediante la interfaz gráfica proporcionada por el servidor web que dispone Jenkins para su gestión.

Cada caso de uso describe los pasos que debe seguir el actor (en este caso, el usuario) para lograr aplicar una determinada funcionalidad, aparte de sus condiciones necesarias.

En este caso, Jenkins es un componente de la solución desarrollado por terceros que permite una gran variedad de casos de uso, debido a sus opciones y complejidad. Por lo tanto, sólo se analizan casos de uso desde un punto de vista global y como un objetivo de la solución con el programa de adaptación Testduino.

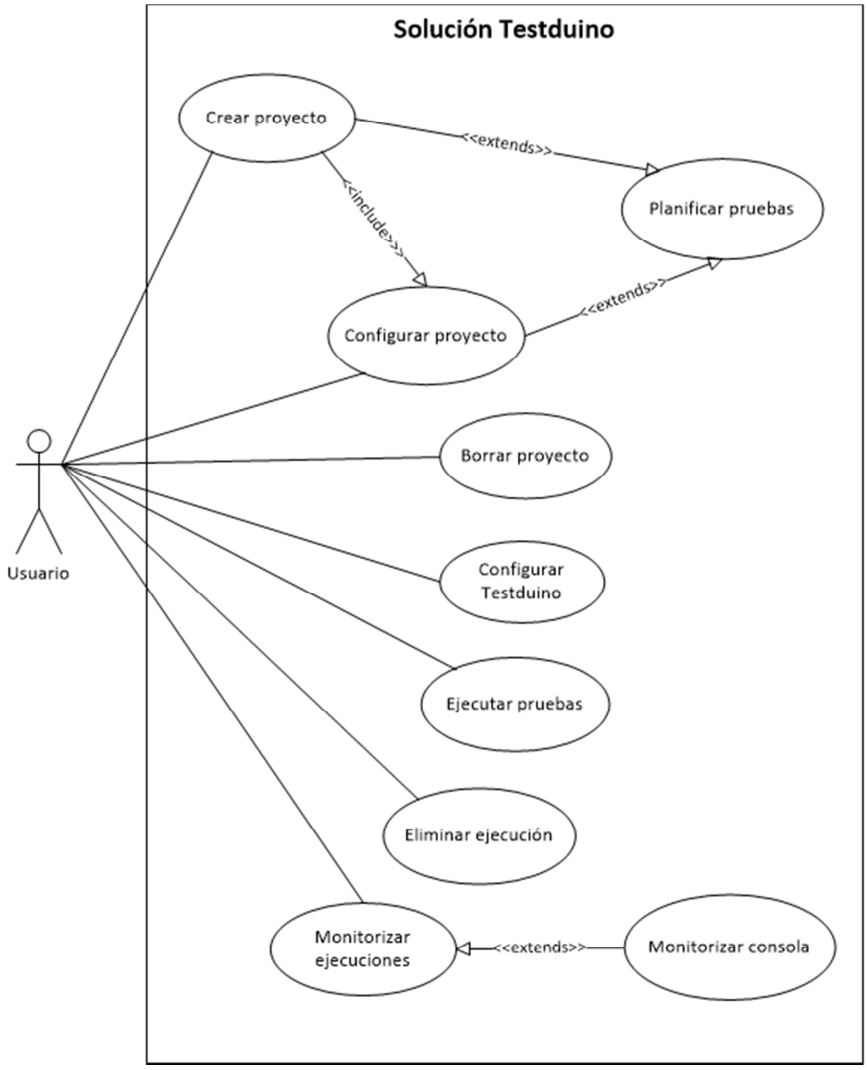


Ilustración 23: Diagrama de casos de uso

A continuación, se describen los casos de uso. La plantilla utilizada para ello es la siguiente:

Identificador:	UC-XX
Nombre:	
Actores:	
Objetivo:	
Descripción:	
Pre-condición:	
Post-condición:	

Tabla 40: Plantilla de caso de uso

- **Identificador:** código único que identifica a cada caso de uso. Se compone de 2 partes:
    - **UC:** corresponde a *Use Case*.
    - **XX:** número de caso de uso.
- **Nombre:** identificación denominando el caso de uso en relación a su función primordial u objetivo. Esta denominación se utiliza en el diagrama de casos de uso para la identificación.
- **Actores:** roles de personas o sistemas que podrán desempeñar una función en el sistema utilizando el caso de uso.
- **Objetivo:** meta final del caso de uso.
- **Descripción:** especificación de los eventos correspondientes a la ejecución estándar del caso de uso.
- **Pre-condición:** especificación de condiciones necesarias para la ejecución del caso de uso.
- **Post-condición:** especificación de condiciones posteriores derivadas de la ejecución del caso de uso.

Identificador:	UC-01
Nombre:	Crear proyecto
Actores:	Usuario
Objetivo:	Establecer un nuevo proyecto de pruebas.
Descripción:	1. En Jenkins, pulsar sobre “Nueva Tarea”. 2. Escribir un nombre para el proyecto. 3. Seleccionar “Crear un proyecto de estilo libre” y pulsar “OK”. 4. Configurar los parámetros que se deseen (opcional). 5. Pulsar “Guardar”.
Pre-condición:	Testduino debe encontrarse disponible. Subversion debe encontrarse disponible.
Post-condición:	El proyecto se encuentra creado con o sin configuración final. Las pruebas se han planificado, si se han configurado.

Tabla 41: UC-01

Identificador:	UC-02
Nombre:	Configurar proyecto
Actores:	Usuario
Objetivo:	Modificar parámetros de configuración y opciones, relativos a la planificación, repositorio de versiones, directorios internos, notificaciones y el comando de Testduino, entre otros permitidos por Jenkins más avanzados.
Descripción:	1. En Jenkins, hacer clic sobre el nombre del proyecto. 2. Pulsar en "Configurar". 3. Configurar los parámetros deseados. 4. Pulsar "Aplicar los cambios" o "Guardar" para salir además.
Pre-condición:	El proyecto ha de encontrarse creado.
Post-condición:	El proyecto actualiza su configuración. El proyecto actualiza su estructura de ficheros.

Tabla 42: UC-02

Identificador:	UC-03
Nombre:	Planificar pruebas
Actores:	Usuario
Objetivo:	Permitir establecer una planificación temporal de pruebas o un modelo de actualización según cambios en el repositorio
Descripción:	1. En Jenkins, hacer clic sobre el nombre del proyecto. 2. Pulsar en "Configurar". 3. En el apartado "Disparadores de ejecuciones", seleccionar la opción deseada. 4. Dependiendo de la opción elegida: a. Si es "Ejecutar después de que otros proyectos se hayan ejecutado", escribir el nombre del proyecto en el campo "*Project names*". b. Si es "Consultar repositorio (SCM)", establecer el programador de consulta en el campo "Programador" según la sintaxis correcta. c. Si es "Ejecutar periódicamente", establecer el programador de ejecuciones en el campo "Programador" según la sintaxis correcta. 5. Pulsar "Aplicar los cambios" o "Guardar" para salir además.
Pre-condición:	El proyecto ha de encontrarse creado y con el repositorio configurado y el comando de Testduino establecido como mínimo.
Post-condición:	El proyecto actualiza su planificación de pruebas. El proyecto actualiza sus ficheros de prueba si existen cambios.

Tabla 43: UC-03

Identificador:	UC-04
Nombre:	Borrar proyecto
Actores:	Usuario
Objetivo:	Eliminar un proyecto, su configuración y sus ficheros relacionados.
Descripción:	
Pre-condición:	El proyecto debe estar creado con o sin configuración final. La ejecución de pruebas debe encontrarse parada o en espera.
Post-condición:	El proyecto se elimina, borrando todos sus ficheros.

Tabla 44: UC-04

Identificador:	UC-05
Nombre:	Configurar Testduino
Actores:	Usuario
Objetivo:	Establecer opciones y parámetros de configuración del programa de adaptación Testduino a través de sus ficheros de configuración o mediante comentarios para Testduino en los *sketches* de prueba.
Descripción:	1. Abrir los ficheros testports.txt y/o testconfig.txt, según la configuración que se desea actualizar, en un editor de texto que soporte texto sin formato. 2. Modificar los ficheros abiertos respetando su estructura e instrucciones. 3. Guardar las modificaciones realizadas. 4. Cerrar el editor de texto.
Pre-condición:	Disponer de un editor de texto sin formato. Disponer de acceso a los ficheros de configuración testports.txt y testconfig.txt. Disponer acceso a los *sketches* de pruebas.
Post-condición:	En la siguiente ejecución, Testduino utiliza la nueva configuración escrita.

Tabla 45: UC-05

Identificador:	UC-06
Nombre:	Ejecutar pruebas
Actores:	Usuario
Objetivo:	Permitir al usuario ejecutar un ciclo de pruebas con Testduino independientemente de la planificación o configuración de ejecución.
Descripción:	1. En Jenkins, hacer clic sobre el nombre del proyecto. 2. Pulsar sobre “Construir ahora”.
Pre-condición:	El proyecto debe estar creado y configurado correctamente con el comando de Testduino y el repositorio de versiones como mínimo.
Post-condición:	Jenkins notifica un resultado al usuario según la configuración del proyecto al finalizar la ejecución del ciclo de pruebas.

Tabla 46: UC-06

Identificador:	UC-07
Nombre:	Cancelar ejecución
Actores:	Usuario
Objetivo:	Detener la ejecución de pruebas en curso.
Descripción:	1. En Jenkins, hacer clic sobre el nombre del proyecto. 2. Pulsar sobre el icono 'X' roja en la ejecución deseada de la lista "Historia de tareas".
Pre-condición:	El proyecto posee al menos una ejecución en proceso en ese mismo instante.
Post-condición:	Jenkins cancela la ejecución. La ejecución pasa a contar como fallida.

Tabla 47: UC-07

Identificador:	UC-08
Nombre:	Eliminar ejecución
Actores:	Usuario
Objetivo:	Eliminar un ciclo de pruebas junto con sus resultados.
Descripción:	1. En Jenkins, hacer clic sobre el nombre del proyecto. 2. Pulsar sobre la ejecución deseada de la lista "Historia de tareas". 3. Pulsar sobre "Borrar esta ejecución".
Pre-condición:	El proyecto se encuentra con al menos una ejecución realizada.
Post-condición:	La ejecución elegida se ha eliminado junto con sus datos. El proyecto tiene en cuenta el número identificativo de la ejecución eliminada.

Tabla 48: UC-08

Identificador:	UC-09
Nombre:	Monitorizar ejecuciones
Actores:	Usuario
Objetivo:	Observar el resultado final de las ejecuciones, así como datos de interés como su fecha y hora, número identificativo, etc.
Descripción:	1. En la página principal de Jenkins, se pueden monitorizar los diferentes proyectos. 2. Al hacer clic sobre el nombre del proyecto, las ejecuciones se observan en la lista "Historia de tareas". Además se pueden controlar los cambios realizados haciendo clic sobre "Cambios".
Pre-condición:	El proyecto ha realizado al menos una ejecución.
Post-condición:	Jenkins muestra al usuario una lista detallada con información sobre cada ejecución realizada.

Tabla 49: UC-09

Identificador:	UC-10
Nombre:	Monitorizar consola
Actores:	Usuario
Objetivo:	Comprobar la salida de consola del programa de adaptación de Testduino, tanto en el momento de ejecución como después de su finalización. Esta salida aporta valiosa información sobre las pruebas y sus trazas de depuración, compilación y carga.
Descripción:	1. En Jenkins, hacer clic sobre el nombre del proyecto. 2. Existen dos formas de realizar el caso de uso: a. Procedimiento 1: i. Pulsar sobre la ejecución deseada de la lista “Historia de tareas”. ii. Pulsar sobre “Salida de consola”. b. Procedimiento 2: i. Pulsar sobre el icono monitor verde que aparece sobre cada ejecución de la lista “Historia de tareas”.
Pre-condición:	El proyecto posee al menos una ejecución realizada o en realización.
Post-condición:	Jenkins muestra al usuario la salida de consola del programa Testduino. En caso de encontrarse en ejecución, la salida se actualiza cada cierto tiempo (no en tiempo real).

Tabla 50: UC-10

3.2.3 Requisitos de software

En este apartado, se muestra la especificación de los requisitos de software, inducidos a partir de los requisitos de usuario mediante un análisis exhaustivo de las funcionalidades requeridas y sus restricciones. Las funcionalidades que se analizan con profundidad corresponden al programa de adaptación, de desarrollo propio, aunque también se detallan ideas implementadas por el software de terceros, especificando este aspecto en todo momento. Sólo se recogen las funcionalidades de los sistemas de terceros relacionadas con la adaptación y su propósito, siendo coherente la idea explicada en los casos de uso.

Los requisitos de software se dividen en funcionales, especificando la funcionalidad del software, y en no funcionales, especificando el procedimiento y la manera de realizar la funcionalidad.

Los requisitos no funcionales se dividen en los siguientes subtipos:

- **Requisitos de interfaz de comunicación:** especifican características y protocolos de comunicación entre componentes de la solución y con el hardware.
- **Requisitos de interfaz hardware:** especifican el tipo de relación entre las placas hardware y su relación con el software de monitorización.
- **Requisitos de interfaz software:** aportan aspectos de la funcionalidad de la solución y su plataforma.
- **Requisitos de interfaz de usuario:** especifican cómo se relaciona el usuario con la solución.
- **Requisitos de portabilidad:** detallan características que permitan la portabilidad de partes de la solución.
- **Requisitos de seguridad:** especifican la metodología de seguridad empleada en la solución.
- **Requisitos de tiempo:** abogan restricciones temporales a la funcionalidad.

El formato seguido en la especificación de los requisitos es el siguiente:

Identificador:	WW-XX-SR-ZZ
Descripción:	
Necesidad:	
Prioridad:	
Estabilidad:	
Origen:	
Grado de verificabilidad:	

Tabla 51: Plantilla de requisito software

A continuación, se describe cada una de las celdas correspondientes a la tabla anterior:

- **Identificador:** es el código único que identifica a cada requisito. Se compone de 4 partes:
    - **WW:** Es F, si el requisito es de software funcional, y NF si el requisito es de software no funcional.
    - **XX:** Solo se aplica para requisitos no funcionales. Los valores permitidos son los siguientes:
        - IC: requisitos de interfaz de comunicación.
        - IH: requisitos de interfaz hardware.
        - IS: requisitos de interfaz software.
        - IU: requisitos de interfaz de usuario.
        - PO: requisitos de portabilidad.
        - SE: requisitos de seguridad.
        - TM: requisitos de tiempo.
    - **SR:** responde a la sigla *Software Requeriment*.
    - **ZZ:** número de requisito.
- **Descripción:** detalla la especificación del requisito de manera precisa y simple.
- **Necesidad:** establece la importancia del requisito desde el punto de vista del cliente. Los valores pueden ser esencial, conveniente u opcional.
- **Prioridad:** establece la importancia del requisito en función del desarrollo del proyecto. Los valores pueden ser alta, media o baja.
- **Estabilidad:** indica el grado de variación que puede experimentar el requisito a lo largo del desarrollo del proyecto. Los valores pueden ser no cambia, cambiante o muy inestable.
- **Origen:** indica la procedencia del requisito: cual o cuales requisitos de usuario lo recogen.
- **Grado de verificabilidad:** mide la dificultad que posee la verificación del cumplimiento del requisito. Los valores pueden ser alta, media o baja.

3.2.3.1 Requisitos funcionales

Identificador:	F-SR-01
Descripción:	Para la integración con el repositorio de versiones y controlar las actualizaciones, el sistema gestiona la funcionalidad de los comandos SVN de Subversion.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-03
Grado de verificabilidad:	Alta

Tabla 52: F-SR-01

Identificador:	F-SR-02
Descripción:	El sistema automatiza el traslado de los *sketches* a su directorio interno (*workspace*) desde los directorios de descarga de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-01 y CA-UR-05
Grado de verificabilidad:	Alta

Tabla 53: F-SR-02

Identificador:	F-SR-03
Descripción:	El sistema automatiza el traslado de las librerías actualizadas al directorio de librerías de Arduino IDE desde el directorio de descarga de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-01 y CA-UR-05
Grado de verificabilidad:	Alta

Tabla 54: F-SR-03

Identificador:	F-SR-04
Descripción:	El sistema permite al usuario la ejecución inmediata de pruebas, incluyendo la actualización a la versión más reciente, a través de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-02
Grado de verificabilidad:	Alta

Tabla 55: F-SR-04

Identificador:	F-SR-05
Descripción:	El sistema ofrece la opción de programar las ejecuciones de pruebas a través de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-02
Grado de verificabilidad:	Alta

Tabla 56: F-SR-05

Identificador:	F-SR-06
Descripción:	El sistema permite al usuario la ejecución de pruebas después de un evento de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-02
Grado de verificabilidad:	Alta

Tabla 57: F-SR-06

Identificador:	F-SR-07
Descripción:	El sistema permite la utilización de opciones avanzadas de ejecución de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-02
Grado de verificabilidad:	Alta

Tabla 58: F-SR-07

Identificador:	F-SR-08
Descripción:	El sistema permite crear y configurar un proyecto de pruebas en Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-05
Grado de verificabilidad:	Alta

Tabla 59: F-SR-08

Identificador:	F-SR-09
Descripción:	El sistema permite configurar cualquier aspecto de un proyecto de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-05
Grado de verificabilidad:	Alta

Tabla 60: F-SR-09

Identificador:	F-SR-10
Descripción:	El sistema permite eliminar proyectos de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-05
Grado de verificabilidad:	Alta

Tabla 61: F-SR-10

Identificador:	F-SR-11
Descripción:	El sistema permite eliminar información sobre ejecuciones realizadas en Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 62: F-SR-11

Identificador:	F-SR-12
Descripción:	El sistema, a través de Jenkins, muestra el estado y mensajes, tanto en tiempo real como después de la ejecución, del programa Testduino.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 63: F-SR-12

Identificador:	F-SR-13
Descripción:	El sistema, a través de Testduino, informa del estado de la configuración, compilación y carga.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 64: F-SR-13

Identificador:	F-SR-14
Descripción:	El sistema, a través de Testduino, muestra los mensajes lanzados por las placas Arduino por puerto serie.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 65: F-SR-14

Identificador:	F-SR-15
Descripción:	El sistema, a través de Testduino, informa al usuario de cualquier error producido en compilación o carga mostrando la salida del compilador o utilidad que ocasione el error.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 66: F-SR-15

Identificador:	F-SR-16
Descripción:	El sistema, en caso de error en la monitorización o el puerto serie, informa al usuario de la imposibilidad de realización del test en concreto.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 67: F-SR-16

Identificador:	F-SR-17
Descripción:	Los errores propios de Testduino se deben informar al usuario.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 68: F-SR-17

Identificador:	F-SR-18
Descripción:	Jenkins notifica al usuario sus propios errores y los del repositorio de versiones, así como los mensajes arrojados por Testduino.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 69: F-SR-18

Identificador:	F-SR-19
Descripción:	En el modo Testduo, Testduino notifica al usuario de los procedimientos realizados de monitorización y los valores a comprobar.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04 y CA-UR-09
Grado de verificabilidad:	Alta

Tabla 70: F-SR-19

Identificador:	F-SR-20
Descripción:	Testduino muestra un resumen de toda la ejecución de pruebas al final de su ejecución.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-04
Grado de verificabilidad:	Alta

Tabla 71: F-SR-20

Identificador:	F-SR-21
Descripción:	El sistema incluye búsqueda recursiva de todos los *sketches* de un directorio base.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-06
Grado de verificabilidad:	Alta

Tabla 72: F-SR-21

Identificador:	F-SR-22
Descripción:	Por cada *sketch* detectado, el sistema gestiona su código de error (si lo hubiere), ruta, modo de ejecución y directorio temporal de ficheros de compilación y carga.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-06, CA-UR-07
Grado de verificabilidad:	Alta

Tabla 73: F-SR-22

Identificador:	F-SR-23
Descripción:	En cada compilación, el sistema administra las librerías de Arduino IDE asignando compilaciones según su utilización por parte de los *sketches*.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-05 y CA-UR-15
Grado de verificabilidad:	Alta

Tabla 74: F-SR-23

Identificador:	F-SR-24
Descripción:	El sistema analiza trazas de TESTDUINO en los comentarios de los *sketches* que indiquen modos de ejecución.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Origen:	CA-UR-07 y CA-UR-15
Grado de verificabilidad:	Alta

Tabla 75: F-SR-24

Identificador:	F-SR-25
Descripción:	El analizador de *sketches* busca expresiones regulares de trazas, añade la cabecera Arduino y gestiona la extensión C++.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Origen:	CA-UR-07, CA-UR-08, CA-UR-10, CA-UR-14 y CA-UR-15
Grado de verificabilidad:	Alta

Tabla 76: F-SR-25

Identificador:	F-SR-26
Descripción:	El sistema ignora cualquier *sketch* que incorpore una traza de modo de ejecución que indique la no ejecución.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	No cambia
Origen:	CA-UR-10 y CA-UR-14
Grado de verificabilidad:	Alta

Tabla 77: F-SR-26

Identificador:	F-SR-27
Descripción:	El sistema obliga la ejecución en un determinado puerto cuando éste esté disponible si se incorpora una traza de modo de ejecución que indique un puerto en concreto en el *sketch*.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Origen:	CA-UR-08 y CA-UR-14
Grado de verificabilidad:	Alta

Tabla 78: F-SR-27

Identificador:	F-SR-28
Descripción:	En caso de detectar el modo de ejecución Testduo, el sistema busca las trazas con los pines y sus valores esperados.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-09, CA-UR-14 y CA-UR-16
Grado de verificabilidad:	Alta

Tabla 79: F-SR-28

Identificador:	F-SR-29
Descripción:	El sistema bloquea la ejecución del *sketch* monitorizado, en el modo Testduo, mientras se ejecuta su comprobación de pines.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-09
Grado de verificabilidad:	Alta

Tabla 80: F-SR-29

Identificador:	F-SR-30
Descripción:	En el modo Testduo, el sistema ejecuta el monitor de la placa monitor reseteándola cada vez que se le requiera.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-09 y CA-UR-11
Grado de verificabilidad:	Alta

Tabla 81: F-SR-30

Identificador:	F-SR-31
Descripción:	En el modo Testduo, el sistema gestiona variaciones en los valores de PWM por errores de hardware.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-09 y CA-UR-16
Grado de verificabilidad:	Alta

Tabla 82: F-SR-31

Identificador:	F-SR-32
Descripción:	Antes de comenzar una prueba, el sistema realiza una prueba del puerto serie a utilizar para saber si está disponible o inutilizable.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-11 y CA-UR-15
Grado de verificabilidad:	Alta

Tabla 83: F-SR-32

Identificador:	F-SR-33
Descripción:	El sistema administra la ejecución de todos los hilos durante un tiempo crítico máximo.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-13
Grado de verificabilidad:	Alta

Tabla 84: F-SR-33

Identificador:	F-SR-34
Descripción:	El sistema gestiona un subproceso o tubería por cada comando, en la consola de comandos de MS-DOS, de compilación o carga.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-15
Grado de verificabilidad:	Alta

Tabla 85: F-SR-34

Identificador:	F-SR-35
Descripción:	El sistema crea y gestiona un hilo de ejecución por cada monitorización, soportando monitorizaciones simultáneas.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-11
Grado de verificabilidad:	Alta

Tabla 86: F-SR-35

Identificador:	F-SR-36
Descripción:	El sistema administra los parámetros necesarios según la configuración de tipo de placa asociada al puerto: UNO o MEGA ADK.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-12 y CA-UR-15
Grado de verificabilidad:	Alta

Tabla 87: F-SR-36

Identificador:	F-SR-37
Descripción:	El sistema se encarga de la gestión del estado de los puertos de comunicación.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-11 y CA-UR-12
Grado de verificabilidad:	Alta

Tabla 88: F-SR-37

Identificador:	F-SR-38
Descripción:	El usuario puede configurar: • Ruta del Arduino IDE. • Ruta de los *sketches* actualizados de Jenkins • Ruta de las librerías actualizadas de Jenkins (opcional). • Ratio de comunicación serie en baudios. • Tiempos de espera máximo (por monitorización y en general). • Palabras de éxito y de error. • Puertos disponibles con el tipo de placa conectada.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	CA-UR-12
Grado de verificabilidad:	Alta

Tabla 89: F-SR-38

3.2.3.2 Requisitos no funcionales

3.2.3.2.1 Requisitos de interfaz de comunicación

Identificador:	NF-IC-01
Descripción:	La ejecución multi-hilo para comunicación se basa en la utilización del módulo de alto nivel *thread* de Python, compatible con Win32.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-01
Grado de verificabilidad:	Alta

Tabla 90: NF-IC-01

Identificador:	NF-IC-02
Descripción:	La ejecución multi-hilo deja disponible cada puerto de comunicación después de su utilización.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-01 y RE-UR-02
Grado de verificabilidad:	Alta

Tabla 91: NF-IC-02

Identificador:	NF-IC-03
Descripción:	Las comunicaciones de mensajes y trazas se componen de caracteres ASCII legibles (números, letras y signos de puntuación como separadores e indicadores).
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-02
Grado de verificabilidad:	Alta

Tabla 92: NF-IC-03

Identificador:	NF-IC-04
Descripción:	En el modo Testduo, la comunicación entre ambas placas Arduino se realiza secuencialmente en un solo hilo de ejecución dedicado a dicha prueba.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-01 y RE-UR-02
Grado de verificabilidad:	Alta

Tabla 93: NF-IC-04

Identificador:	NF-IC-05
Descripción:	En el modo Testduo, el orden de la comunicación es: 1. Comunicación con el Arduino monitorizado. 2. Comunicación con el Arduino monitor. 3. Si procede, comunicación con el Arduino monitorizado para continuar con la ejecución del *sketch*.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-02
Grado de verificabilidad:	Alta

Tabla 94: NF-IC-05

Identificador:	NF-IC-06
Descripción:	El *sketch* de Arduino, obligatoriamente, utiliza, para la comunicación, la librería Serial estándar incluida en el API de Arduino.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-02
Grado de verificabilidad:	Alta

Tabla 95: NF-IC-06

Identificador:	NF-IC-07
Descripción:	La comunicación serie estándar es de 9600 baudios.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-02 y RE-UR-06
Grado de verificabilidad:	Alta

Tabla 96: NF-IC-07

Identificador:	NF-IC-08
Descripción:	La orden de ejecución de Testduino, por parte de Jenkins, consiste en la ejecución de un comando con el intérprete de Python.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-03
Grado de verificabilidad:	Alta

Tabla 97: NF-IC-08

Identificador:	NF-IC-09
Descripción:	El programa Testduino retorna 0 en caso de éxito total o -1 si ha existido algún error mediante la *shell* de MS-DOS a Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-03
Grado de verificabilidad:	Alta

Tabla 98: NF-IC-09

Identificador:	NF-IC-10
Descripción:	El sistema, a través de Jenkins, utiliza un cliente de protocolo SVN o SVN+SSH para la comunicación, sobre TCP/IP, con el repositorio de versiones.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-04
Grado de verificabilidad:	Alta

Tabla 99: NF-IC-10

3.2.3.2.2 Requisitos de interfaz hardware

Identificador:	NF-IH-01
Descripción:	La configuración y lectura de pulsos se realiza con la API de Arduino para el manejo de los pines digitales E/S del microcontrolador.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-05
Grado de verificabilidad:	Alta

Tabla 100: NF-IH-01

Identificador:	NF-IH-02
Descripción:	En las trazas, los pines siguen la misma numeración del API de Arduino, soportando hasta el pin digital 50.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-05
Grado de verificabilidad:	Alta

Tabla 101: NF-IH-02

Identificador:	NF-IH-03
Descripción:	Si se detecta un problema en un puerto serie del Arduino, se considera error en la prueba.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-06
Grado de verificabilidad:	Alta

Tabla 102: NF-IH-03

Identificador:	NF-IH-04
Descripción:	En caso de no encontrar ningún puerto disponible con placa Arduino, la prueba se da por fallida.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-07
Grado de verificabilidad:	Alta

Tabla 103: NF-IH-04

3.2.3.2.3 Requisitos de interfaz software

Identificador:	NF-IS-01
Descripción:	El programa Testduino se ejecuta sobre la consola de MS-DOS o Windows PowerShell.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-08
Grado de verificabilidad:	Alta

Tabla 104: NF-IS-01

Identificador:	NF-IS-02
Descripción:	El programa Testduino se ejecuta sobre Windows 7 y Windows 8, tanto en 32 bits como en 64 bits.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-08
Grado de verificabilidad:	Alta

Tabla 105: NF-IS-02

Identificador:	NF-IS-03
Descripción:	El programa Testduino se ha implementado adecuadamente para su interpretación con Python 3.x de 32 bits.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-09
Grado de verificabilidad:	Alta

Tabla 106: NF-IS-03

Identificador:	NF-IS-04
Descripción:	El programa Testduino utiliza las librerías PySerial y PyWin32, ambas de 32 bits, para Python 3.x.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-09
Grado de verificabilidad:	Alta

Tabla 107: NF-IS-04

Identificador:	NF-IS-05
Descripción:	Jenkins puede ejecutarse sobre Apache Tomcat o como servicio de Windows.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-08
Grado de verificabilidad:	Alta

Tabla 108: NF-IS-05

Identificador:	NF-IS-06
Descripción:	El sistema muestra su interfaz, perteneciendo a Jenkins, a través de navegadores web que cumplan los estándares W3C.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-08
Grado de verificabilidad:	Alta

Tabla 109: NF-IS-06

Identificador:	NF-IS-07
Descripción:	Los ficheros de configuración de Testduino son de texto plano y siguiendo la extensión TXT.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-10
Grado de verificabilidad:	Alta

Tabla 110: NF-IS-07

Identificador:	NF-IS-08
Descripción:	El programa Testduino utiliza la configuración del fichero avrdude.conf del IDE de Arduino para el programa de carga avrdude.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-11
Grado de verificabilidad:	Alta

Tabla 111: NF-IS-08

Identificador:	NF-IS-09
Descripción:	Los pasos de la compilación y carga son: 1. Inclusión de la cabecera <Arduino.h> en el *sketch*. 2. Analizado del *sketch* para la búsqueda de librerías. 3. Compilación de *sketch* y librerías. 4. Enlazado de ficheros objeto, optimización y generación del hexadecimal. 5. Carga del hexadecimal.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-11
Grado de verificabilidad:	Alta

Tabla 112: NF-IS-09

Identificador:	NF-IS-10
Descripción:	Para una correcta compilación, los *sketches* deben incorporar la definición de funciones antes de su implementación, según el estándar de ANSI C y Processing.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-11
Grado de verificabilidad:	Alta

Tabla 113: NF-IS-10

Identificador:	NF-IS-11
Descripción:	En el modo Testduo, en los puntos de ruptura, el bloqueo del *sketch* a monitorizar no realiza operaciones.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-12
Grado de verificabilidad:	Alta

Tabla 114: NF-IS-11

Identificador:	NF-IS-12
Descripción:	La secuencia del modo Testduo se compone de: 1. Bloquear el *sketch* y enviar traza con valores estimados. 2. Testduino envía traza a Arduino monitor con pines a monitorizar. 3. Devolver traza enviada con valores reales de los pines. 4. Si coinciden, enviar instrucción de continuación, en otro caso, error.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-12
Grado de verificabilidad:	Alta

Tabla 115: NF-IS-12

3.2.3.2.4 Requisitos de interfaz de usuario

Identificador:	NF-IU-01
Descripción:	El panel de administración de Jenkins es accesible desde una dirección URL.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-13
Grado de verificabilidad:	Alta

Tabla 116: NF-IU-01

Identificador:	NF-IU-02
Descripción:	La interacción con el programa Testduino se limita a su interrupción por parte del usuario.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-14
Grado de verificabilidad:	Alta

Tabla 117: NF-IU-02

Identificador:	NF-IU-03
Descripción:	El programa Testduino identifica cada línea de consola, para saber a qué se refiere el mensaje, al comienzo de la línea.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-15
Grado de verificabilidad:	Alta

Tabla 118: NF-IU-03

3.2.3.2.5 Requisitos de portabilidad

Identificador:	NF-PO-01
Descripción:	El programa Testduino se puede ejecutar manualmente sin necesidad de Jenkins.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-16
Grado de verificabilidad:	Alta

Tabla 119: NF-PO-01

Identificador:	NF-PO-02
Descripción:	El programa Testduino consiste en ficheros .py de Python cuya instalación se basa en la operación copiar y pegar.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-16
Grado de verificabilidad:	Alta

Tabla 120: NF-PO-02

3.2.3.2.6 Requisitos de seguridad

Identificador:	NF-SE-01
Descripción:	La generación de objetos compilados y ficheros auxiliares se realiza en un directorio temporal ubicado en la misma ruta del *sketch*.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-17
Grado de verificabilidad:	Alta

Tabla 121: NF-SE-01

Identificador:	NF-SE-02
Descripción:	El sistema trabaja con un espacio de trabajo propio copiando los *sketches* para no alterar otros ficheros válidos.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-17
Grado de verificabilidad:	Alta

Tabla 122: NF-SE-02

3.2.3.2.7 Requisitos de tiempo

Identificador:	NF-TM-01
Descripción:	La unidad de medición de tiempos del programa Testduino es el segundo.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-18
Grado de verificabilidad:	Alta

Tabla 123: NF-TM-01

Identificador:	NF-TM-02
Descripción:	La regulación temporal de la monitorización por cada puerto serie se realiza mediante un *timeout* configurable.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-18
Grado de verificabilidad:	Alta

Tabla 124: NF-TM-02

Identificador:	NF-TM-03
Descripción:	La regulación temporal del programa Testduino, para evitar bloqueos de proceso, se realiza mediante un *timeout* general configurable.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Origen:	RE-UR-18
Grado de verificabilidad:	Alta

Tabla 125: NF-TM-03

4. Diseño de la solución

En este apartado, se detalla el diseño realizado según la arquitectura empleada, así como todos sus elementos hardware y el software necesario para la integración continua de Arduino.

Es importante recalcar que Jenkins, Subversion, las herramientas AVR de carga y compilación y las plataformas empleadas actúan como cajas negras desde nuestra perspectiva, ya que han sido desarrolladas por terceros y no es necesaria ninguna modificación en ellas. Aun así, se detallará su funcionamiento y diseño, así como su relación con el resto de componentes de la solución.

En el apartado dedicado al hardware, se analizarán los procedimientos necesarios empleados por Testduino, así como el diseño detallado del software empotrado del modo Testduo, gestionado por el microcontrolador. Por otra parte, en el apartado dedicado al software, se realiza el diseño detallado del programa Testduino.

4.1 Hardware y software empotrado

A continuación, se describe brevemente la plataforma de open hardware Arduino, su diseño y funcionamiento, y se incide más específicamente en los aspectos relevantes a la solución. También, se especifica el funcionamiento del software empotrado necesario y sus comunicaciones.

Arduino es una plataforma open hardware, basada en un microcontrolador Atmel de arquitectura ATMEGA que permite realizar operaciones de cómputo y control, limitadas por su velocidad y capacidad de memoria. Arduino, como plataforma, proporciona un API de funciones, basadas en las estándares de los lenguajes, y librerías para facilitar la programación del microcontrolador, que se realiza oficialmente en una variación del lenguaje C/C++ muy similar al estándar utilizado en los compiladores GNU GCC, derivada del lenguaje de microcontroladores Processing. Arduino es la parte hardware de la solución, sobre el cual, se ejecutan las pruebas programadas.

4.1.1 Objetivo

El objetivo de cada una de las placas Arduino conectadas, es la simple ejecución de los *sketches* de las pruebas y posibilitar la monitorización mediante la comunicación serie con el equipo, además de realizar otro tipo de comunicaciones pertinentes de que el *sketch* lo requiera siempre que se cuente con el *shield* adecuado. En el modo Testduo, se añade el objetivo de que un *sketch* monitor se comunique con el *sketch* que ejecuta la prueba y le monitorice su E/S de pines digitales (SPI), y aparte de esto, el *sketch* monitor debe enviar los resultados de E/S al equipo para su posterior análisis.

4.1.2 Funcionamiento

En esta sección, se describe el funcionamiento de Arduino a nivel de hardware y a nivel de software empotrado. El desarrollo de librerías y *sketches* para el modo Testduo se comenta posteriormente en la sección de software.

Arduino es, simplemente, un microcontrolador acompañado de su correspondiente electrónica necesaria para su funcionamiento. El microcontrolador nos provee de funciones de computación y control con un cierto límite, ya que está pensado para un consumo muy bajo de energía y un bajo coste, y esto acarrea ciertos límites de memoria y velocidad de cómputo.

Los microcontroladores Arduino son marca Atmel, de la familia ATMEGA. Se suelen incluir dos: el principal y uno dedicado a la traducción FTDI a USB del UART dedicado a la comunicación con el equipo. En ciertos modelos de ATMEGA, el propio microcontrolador principal ya incorpora ésta característica, siendo innecesario un segundo microcontrolador.

Las características de los microcontroladores ATMEGA soportados por Testduino son las siguientes:

Microcontrolador	ATMEGA328	ATMEGA2560
Placa Arduino	Arduino UNO	Arduino Mega ADK
Voltaje de operación	5 V	5 V
E/S digital	14 (6 PWM)	54 (15 PWM)
Entradas analógicas	6	16
Memoria *flash*	32 KB	256 KB
Memoria de *bootloader*	0,5 KB	8 KB
SRAM	2 KB	8 KB
EEPROM	1 KB	4 KB
Velocidad de reloj	16 MHz	16 MHz

Tabla 126: Especificaciones técnicas de las placas Arduino compatibles

El software empotrado que proporciona la lógica al microcontrolador consta de dos partes:

- ***Bootloader***: viene por defecto cargado en cualquier Arduino, aunque es reprogramable desde el entorno Arduino IDE por conexión directa mediante FTDI. Cada modelo de placa Arduino posee uno propio y proporciona la secuencia de arranque y de estados de la E/S del microcontrolador. Está programado en ensamblador y C. No necesita modificación para el funcionamiento de Testduino.

- ***Sketch***: es la lógica programada que determina el funcionamiento del microcontrolador. Se programan en lenguaje de alto nivel tipo C/C++ o Python, que posteriormente se traduce al hexadecimal correspondiente para su carga en el microcontrolador. Es compatible con todas las placas Arduino utilizando la librería Arduino.h proporcionada con el IDE.

A continuación, se describe en detalle el proceso de compilación y carga del software empotrado utilizado en este proyecto.

4.1.3 Compilación

En este proyecto, se utilizan diversos *sketches* que se comportan como *testers* de funciones de librerías o hardware externo. Además, para el modo Testduo se requiere de un *sketch* especial de monitorización. Cada uno de estos *sketches* debe ser compilado y cargado, en su momento, en la placa Arduino correspondiente.

Los *sketches* pueden ser programados en diferentes lenguajes de alto nivel, si bien, los oficiales soportados por el entorno Arduino IDE y sus herramientas de compilación son C/C++. Testduino utiliza las mismas herramientas de compilación del IDE oficial de la misma forma, igualando el proceso del IDE. De esta forma, podemos asegurar que el resultado de la ejecución del *sketch* será el mismo, ya que el proceso es largo y requiere de múltiples pasos, en los cuales, cualquier alteración puede ocasionar un comportamiento no deseado o incluso errores de memoria o de enlace de objetos, lo que repercute en una alteración grave del fichero hexadecimal final. Por este motivo, es necesaria la instalación del IDE oficial de Arduino, así como la posterior configuración de su ruta absoluta en Testduino.

Las herramientas para la compilación utilizadas forman parte del paquete AVR-GCC destinado a ello. Incluye diversos comandos para la generación de los ficheros compilados y obtener el binario hexadecimal correspondiente. Detalladamente son:

- **GNU GCC (avr-gcc) y GNU G++ (avr-g++):** compiladores de C y C++ respectivamente pertenecientes al sistema operativo libre GNU, basados en el clásico GNU GCC y adaptados para la generación del compilado compatible con la arquitectura AVR de los microcontroladores ATMEGA. Sus objetivos son la compilación del *sketch*, sus librerías asociadas y las librerías básicas de Arduino. A continuación, analizamos los parámetros utilizados[50]:
    - **-c:** compila los ficheros de código fuente sin enlazarlos, simplemente crea un fichero objeto por cada fichero de código fuente.
    - **-g:** produce información de depuración en el formato nativo del sistema operativo (*stabs*, COFF, XCOFF o DWARK 2).
    - **-Os:** optimiza el espacio mediante optimizaciones -O2 sin incrementar el tamaño del código.
    - **-Wall:** muestra todas las advertencias de compilación.
    - **-fno-exceptions:** deshabilita las excepciones de bloques try/catch.
    - **-ffunction-sections:** reubica cada función en su propia sección del enlazador, que permite al enlazador mejorar la extracción de código muerto, optimizando el *kernel* resultante.
    - **-fdata-sections:** coloca cada ítem de datos o función en su propia sección en el fichero de salida, siempre que el destino final soporte secciones arbitrarias.
    - **-mmcu=:** especifica el set de instrucciones AVR o el tipo de MCU.
    - **-DF_CPU=:** frecuencia del microcontrolador AVR.

- **-MMD:** gestiona las dependencias en el fichero principal.
- **-DUSB_VID=:** Identificación del fabricante para dispositivos USB.
- **-DUSB_PID=:** Identificación del producto Arduino como dispositivo USB.
- **-DARDUINO=:** Directiva del software de programación Arduino IDE.
- **-I:** incorpora las rutas de directorios con los ficheros de cabecera necesarios.
- **-o:** incorpora las rutas de los ficheros objeto de salida.
- **-Wl,--gc-sections,--relax:** reduce el tamaño final del *kernel* permitiendo al enlazador relajar las llamadas a funciones cercanas.
- **-L:** incorpora las rutas de directorios necesarias para las cabeceras especificadas en -l.

- **GNU AR (avr-ar):** denominado archivador, fue construido para los microcontroladores AVR especialmente. Su función es comprimir y agregar objetos compilados en una librería única y crear un índice para que el enlazador pueda utilizarla. La librería que se crea en nuestro proceso se denomina core.a e incluye todas las librerías utilizadas, tanto las añadidas por el *sketch* como las básicas, en este único fichero. Los parámetros utilizados son[51]:
  - **r:** ordena la inserción de módulos objeto en el fichero de salida.
  - **c:** crea un contenedor.
  - **s:** crea un índice del contenedor.

- **GNU OBJCOPY (avr-objcopy):** su función es copiar el contenido de un fichero contenedor de un objeto a otro. Utiliza la librería GNU BFD para escribir y leer ficheros que contienen objetos, reconociendo la mayoría de los formatos estándares actuales. En Arduino, es utilizado para transformar el fichero binario ELF, generado por el compilador GNU GCC para el *sketch* y la librería core.a, a un fichero EEP de sección de EEPROM y generar el definitivo binario hexadecimal. En detalle, los parámetros necesarios son[52]:
  - **-O:** escribe el fichero de salida en el formato especificado.
  - **ihex:** formato hexadecimal.
  - **-j:** copia sólo las secciones con denominación en el fichero de salida.
  - **--set-section-flags=.eeprom=alloc,load:** establece *flags* en la sección especificada.
  - **--no-change-warnings:** no informa de advertencias de secciones no denominadas o de cambios en ellas.

- **--change-section-lma:** establece o cambia la dirección LMA de la sección especificada.

- **-R:** elimina cualquier sección especificada del fichero de salida.

Las herramientas AVR-GCC, al pertenecer al sistema operativo GNU, están desarrolladas para plataformas UNIX compatibles. Con lo cual, la versión definitiva utilizada tanto por el IDE oficial (versión Windows) como por Testduino es una adaptación realizada para sistemas Microsoft denominada WinAVR, que incluye las mismas herramientas en un formato EXE de ejecución desde consola de MS-DOS o Windows PowerShell. Dichas herramientas adaptadas requieren utilizar la librería cygwin1.dll, destinada a la emulación de UNIX en plataformas Windows, incluida en el IDE oficial de Arduino.

La compilación requiere la utilización de las herramientas descritas en un orden correcto y con unos parámetros determinados.

A continuación, se describen los pasos de compilación, realizados por Testduino, detalladamente, acompañados de un diagrama ilustrativo:

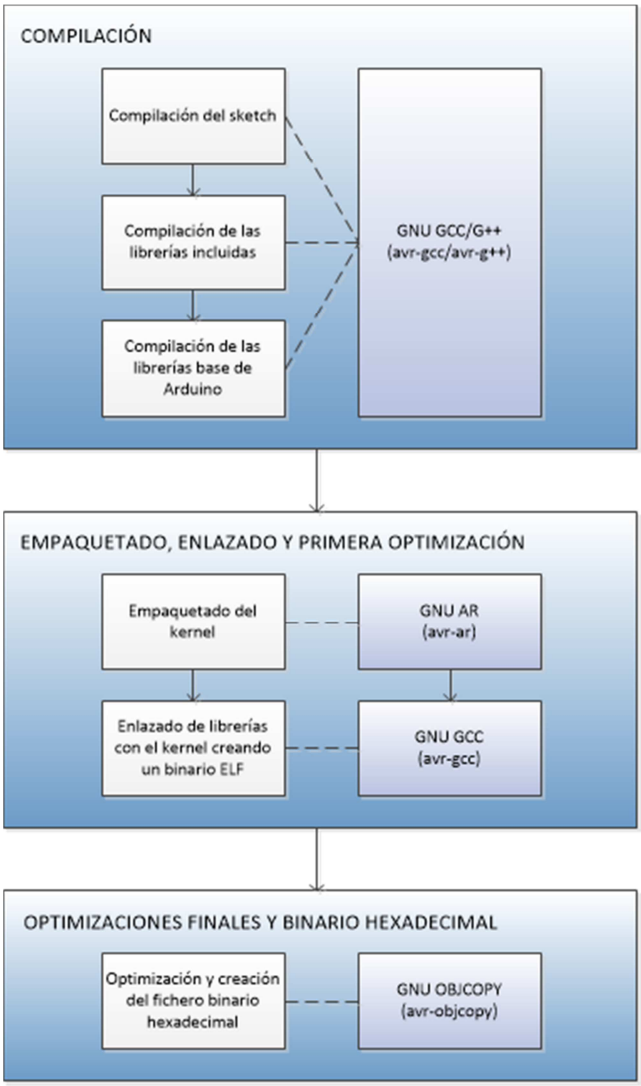


Ilustración 24: Esquema de compilación

1. **Compilación del *sketch*:** inicialmente, el *sketch*, representado por un fichero INO, se transforma en un fichero CPP de C++ añadiéndole la directiva de inclusión de la librería base Arduino.h. Posteriormente, se compila con el compilador GNU G++ proporcionándole las rutas de las librerías utilizadas en el *sketch* y las librerías base de Arduino, así como información sobre el hardware (frecuencia, modelo de microcontrolador, identificadores USB, etc.), sobre el cual, se ejecutará. Se intenta optimizar el código anulando soporte de excepciones y relajando enlaces de funciones y secciones.

2. **Compilación de las librerías incluidas:** las librerías incluidas por el usuario en el *sketch* serán compiladas utilizando el compilador GNU G++ si están escritas en C++ o el compilador GNU GCC si están escritas en C. Se proporcionan las rutas necesarias a los compiladores sobre dependencias y librerías base de Arduino, además de información sobre el hardware destino. Se intenta optimizar el código anulando soporte de excepciones y relajando enlaces de funciones y secciones.

3. **Compilación de las librerías base de Arduino**: escritas en C y C++, proveen la funcionalidad básica e información necesaria para el correcto funcionamiento del *sketch* sobre el hardware. Se compilan con GNU GCC o GNU G++ según sea el caso, proporcionándoles información sobre el hardware destino, optimizando el código anulando excepciones y relajando enlaces de funciones y secciones.

4. **Empaquetado del kernel:** las librerías base de Arduino y el *sketch* componen el *kernel*, *core* o núcleo del sistema, que se empaqueta en un fichero denominado core.a mediante GNU AR. Este archivador se ejecuta por cada librería compilada sobre su fichero objeto, y lo adjuntará al fichero kernel.

5. **Enlazado de librerías con el kernel:** utilizando el compilador GNU GCC se crea un fichero binario de formato ELF que incorpore las librerías incluidas por el usuario al kernel creado en el paso anterior. Se proporciona información sobre el microcontrolador, así como parámetros de optimización y relajación de vínculos entre funciones.

6. **Optimización y creación del fichero binario hexadecimal:** este último paso consiste en dos ejecuciones de la herramienta GNU OBJCOPY con dos propósitos.

    a. ***Flags* y direcciones con fichero EEP:** se establecen las *flags* de *alloc* y *load* en la sección referente a la memoria EEPROM, además de cambiar su dirección LMA a 0. Se genera un fichero resultante EEP.

    b. **Generación del fichero final HEX:** eliminando secciones no imprescindibles, transforma el binario ELF en un binario hexadecimal, el cual, se cargará posteriormente en la placa Arduino.

4.1.4 Carga

El proceso de carga de Arduino consiste en escribir la información hexadecimal obtenida en el proceso de compilación, en un fichero de formato HEX, en la memoria del microcontrolador.

La carga se realiza mediante conexión USB serie, y para ello, es necesario disponer de un microcontrolador que actúe como traductor FTDI-Serie e interactúe con el UART del microcontrolador principal destinado a ello. Afortunadamente, todas las placas oficiales de Arduino cuentan con este soporte ya integrado tanto a nivel de hardware como de software.

Por otra parte, el microcontrolador principal debe disponer de un *bootloader* cargado previamente, este *bootloader* le indica el procedimiento inicial de carga dentro del chip, así como su inicio de ejecución. En Testduino, se obvia que el *bootloader* se encuentra cargado, ya que así vienen por defecto las placas oficiales Arduino, no siendo así en los chips Atmel adquiridos directamente en la propia Atmel. La carga por programador a través de FTDI no está soportada por Testduino.

La carga requiere de un solo paso: iniciar la herramienta AVRDUDE. AVRDUDE es un programa para cargar código y datos en cualquier microcontrolador AVR. Soporta varios programadores, además de la programación mediante comunicación serie. La herramienta recibe como entradas el fichero HEX y el fichero de configuración avrdude.conf, éste último incluido en el IDE oficial de Arduino, tomado también por Testduino. Los parámetros utilizados en la ejecución son[53]:

- **-C:** especifica la ruta del fichero de configuración, en nuestro caso, el fichero avrdude.conf.
- **-p:** indica el tipo de microcontrolador a programar. Es el único parámetro obligatorio en cada invocación.
- **-c:** indica el identificador del programador a utilizar para el microcontrolador.
- **-P:** especifica el puerto serie dedicado para la carga.
- **-b:** especifica la tasa de baudios empleada en la carga.
- **-D:** deshabilita el auto-borrado de la memoria *flash* del microcontrolador.
- **-Uflash:w:[ruta_fichero_a_cargar]:i:** indica que se debe realizar una lectura del fichero indicado y escribirlo en la memoria *flash* del microcontrolador. La letra 'i' indica que el fichero con el programa se encuentra en formato Intel Hex.

  Una vez cargado el programa en el microcontrolador, su ejecución es inmediata.

4.1.5 Software empotrado

4.1.5.1 Testduino Single

La integración con el resto de elementos de la solución se realiza a través de la comunicación con Testduino mediante el puerto serie. Para ello, es imprescindible que los *sketches* de prueba incluyan la apertura del puerto a 9600 baudios y realicen las trazas de depuración deseadas enviando dicha información por el puerto mencionado. Para ello, se deberán utilizar, en el código del *sketch*, las siguientes funciones de la clase Serial:

- **Serial.begin(int baudrate):** inicia la comunicación serie manteniendo la tasa de baudios especificada por parámetro.

- **Serial.print(val):** imprime por el puerto serie el parámetro val, pudiendo éste ser cualquier tipo de dato (enteros, *string*, *arrays*, etc.).

- **Serial.println(val):** aplica el mismo funcionamiento que Serial.print(val), añadiendo un retorno de carro final.

- **Serial.write(val):** imprime datos binarios por el puerto serie. Puede ser un *byte* simple o un *array* de bytes.

- **Serial.read():** lee la información recibida a través del puerto serie.

- **Serial.readBytes(buffer, int length):** lee *bytes* recibidos a través del puerto serie y los almacena en un *buffer* de tipo *char* o *byte*.

- **Serial.flush():** espera a que cualquier transmisión serie termine para limpiar la comunicación.

Existen otras sobrecargas con otros parámetros y otras funciones diferentes, pero no son muy relevantes para la comunicación necesaria. Por otra parte, en el Arduino MEGA ADK existen varios puertos serie de comunicación, a los cuales se deben referir con Serial1, Serial2 o Serial3 dependiendo de cuál de ellos se encuentre conectado al equipo.

El equipo actúa de forma pasiva, recibiendo la información y buscando cualquier traza de error especificada, o de éxito.

4.1.5.2 Testduo

En el modo Testduo, la integración es aún mayor y la comunicación más compleja. La monitorización de ambas placas Arduino requiere la misma comunicación descrita de manera duplicada. Por supuesto, la interacción entre placas Arduino requiere del uso de un protocolo entre ellas y con el equipo, y el envío de trazas con resultados esperados requiere de una librería para facilitar su uso desde cualquier *sketch*. Testduo sólo es compatible con placas tipo Arduino UNO.

4.1.5.2.1 Comunicación entre placas

La comunicación entre placas se realiza a dos niveles: hardware y software. A nivel hardware, es necesaria la unión física de los pines a monitorizar con los mismos pines en la placa Arduino monitor. A nivel software, la placa Arduino monitor posee cargado un *sketch* especial de monitorización que realiza las lecturas pertinentes y enviarlas al equipo, que las analizará en comparación con la traza de resultado esperado recibida desde la placa Arduino monitorizada.

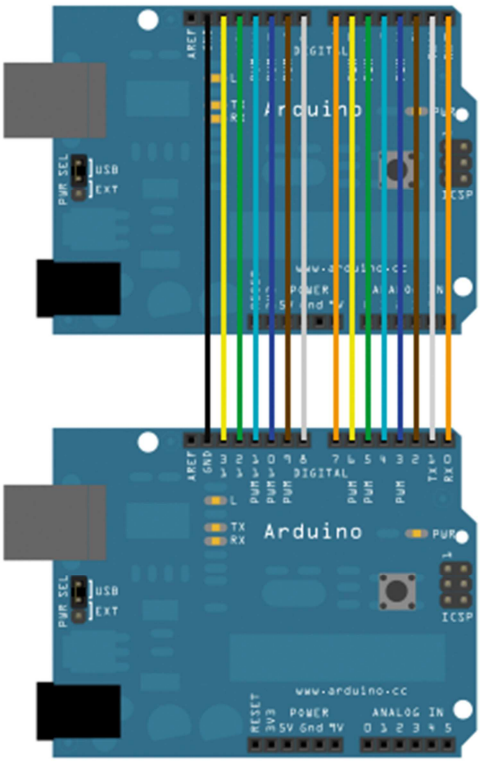


Ilustración 25: Esquema de conexión total de pines

El *sketch* especial de monitorización realiza una lectura del valor que transmite el pin de la placa monitorizada y lo enviará por el puerto serie al equipo. La lectura puede ser digital o para salidas PWM, automáticamente se realiza la apropiada según el valor. En el caso de la lectura digital, se emplea la función *digitalRead(uint8_t pin)* de Arduino, apropiada para ello.

La lectura para salidas PWM es un poco más compleja. Arduino permite salidas de PWM, estas salidas son pulsos de duración controlada que simulan salidas analógicas con una salida digital. Estos pulsos son de 5V con una frecuencia, en los microcontroladores Atmel, de alrededor de 500 Hz, lo que equivale a periodos de 2 milisegundos. La API de Arduino utiliza un rango de enteros de entre 0 y 255 para configurar el periodo de los pulsos, siendo 255 el máximo o equivalente a una salida digital a 1 y 0 el mínimo o equivalente a una salida digital a 0. En la lectura de valores PWM se debe analizar el periodo del pulso entrante y calcular su equivalencia en el rango 0-255.

La lectura se realiza utilizando la función *pulseIn(uint8_t pin, int value)*, que proporciona una lectura de un pulso entrante y devuelve su longitud en microsegundos. El parámetro *value* indica el tipo de pulso a leer, que puede ser alto *(5V, HIGH)*, como es nuestro caso, o bajo *(0V, LOW)*.

El flujo de ejecución principal del *sketch* es el siguiente:

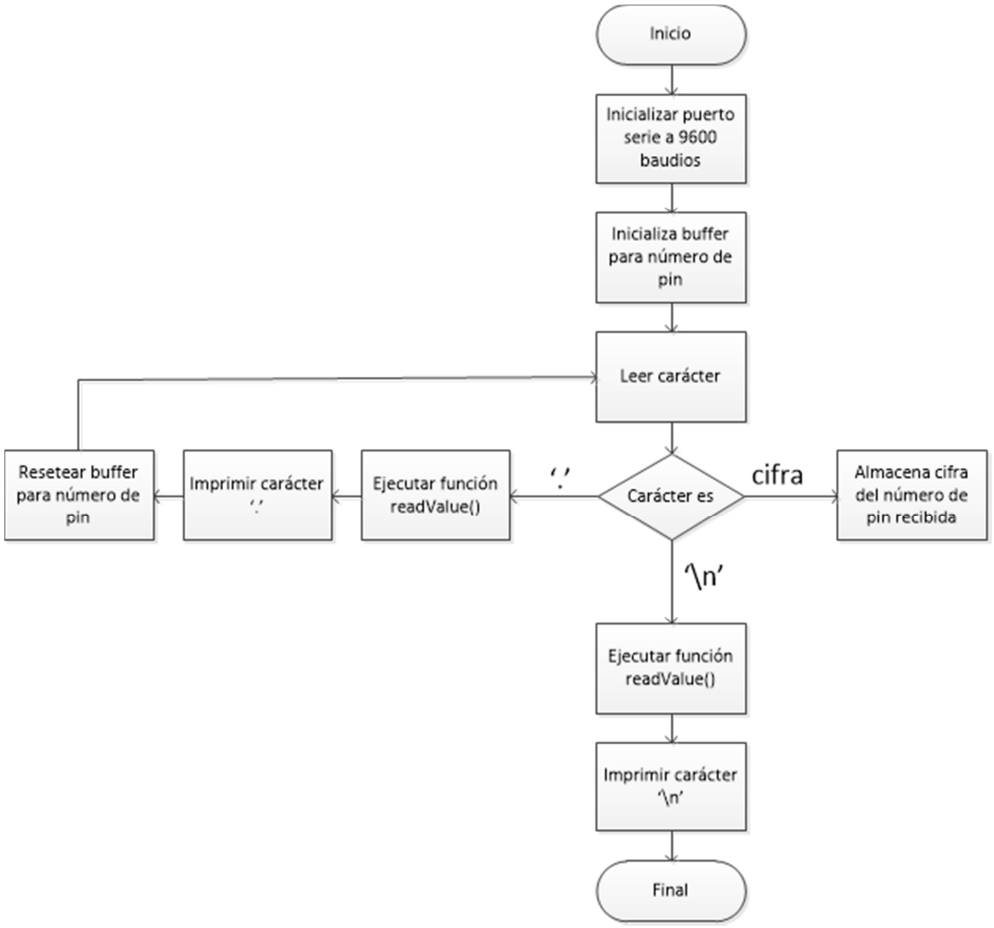


Ilustración 26: Diagrama de flujo principal del sketch de monitorización

Como se puede observar, el funcionamiento del *sketch* se divide en la función principal (*setup()*, donde se inicializan las variables, y *loop()*, donde se realizan las operaciones) y en la función *readValue()*.

La función principal, mayoritariamente *loop()*, lee carácter a carácter del puerto serie una cadena que incluye cada número de pin a monitorizar, separados por un punto, y terminada en un retorno de carro. Un ejemplo sería:

2.3.5.10.13\n → Esta cadena ordena leer los valores de los pines 2, 3, 5, 10 y 13.

La cadena proviene del equipo, que la ha generado a través de la traza que le mandó el Arduino a monitorizar. Una vez leídos los pines, se devolverá al equipo por el mismo puerto serie otra cadena que indica el valor de cada pin en la misma posición en la que se encuentra dicho número de pin en la cadena recibida. Siguiendo el ejemplo anterior, una posible respuesta sería:

234.1.0.12.1\n → Esta cadena indica los siguientes valores para los siguientes pines:

- Para el pin 2, un valor de PWM aproximado de 234.
- Para el pin 3, un valor digital de 1 o de PWM aproximado de 1.
- Para el pin 5, un valor digital de 0 o de PWM aproximado de 0.
- Para el pin 10, un valor de PWM aproximado de 12.
- Para el pin 13, un valor digital de 1 o de PWM aproximado de 1.

Los valores digitales pueden ser 0 o 1, y los valores PWM se devuelven en el rango 0-255 manejado por la API de Arduino. Un valor 0 de PWM es lo mismo que un valor digital 0, pero esto no se cumple con un valor 1 de PWM, con lo que, Testduino sólo tiene en cuenta el valor en sí y no si es digital o de PWM, ya que una escritura analógica no puede producir un 1 digital, lo que sería un 255 en PWM, y una escritura digital nunca puede producir un valor mayor que 1.

El funcionamiento de la lectura es sencillo. Como máximo los números de pines pueden tener 2 cifras, entonces, se almacenan las cifras recibidas hasta que se recibe un punto, ya que en ese caso tendremos un número de pin completo y llamaremos a *readValue()*. Por otra parte, si se recibe un retorno de carro, la cadena ha finalizado. Se puede deducir que el *sketch* lee y responde los valores al tiempo que recibe los números de los pines.

La lectura de los valores la realiza la función auxiliar *void readValue()*, cuyo funcionamiento consiste en tomar el número de pin almacenado y practicar las funciones de lectura del API de Arduino para obtener el valor transmitido por el Arduino monitorizado y enviarlo por el puerto serie al equipo. Su diagrama de flujo es el siguiente:

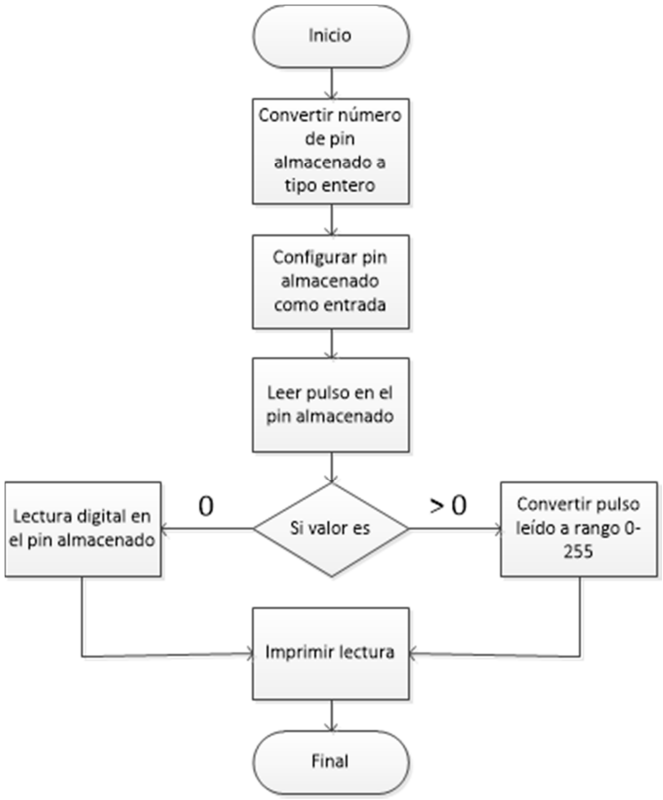


Ilustración 27: Diagrama de flujo de la función auxiliar readValue()

Según las diferentes pruebas realizadas, la reacción al leer un pulso de una escritura digital con el API de Arduino es siempre cero, de ahí la comparación que se establece para comprobar si se trata de un pulso PWM o de un valor digital. Por otra parte, la transformación del valor leído en microsegundos al rango PWM 0-255 se realiza dividiendo el valor leído entre 8, salvo si nos encontramos monitorizando los pines 5 y/o 6, en los cuales, el valor leído debe dividirse por 4, ya que la frecuencia de los pulsos es diferente por estar gestionados por otro *timer* del microcontrolador[54]. El valor PWM obtenido final no es exacto, debido a los errores de precisión que pueden afectar por parte del hardware del microcontrolador y del API de Arduino, y puede variar ±3 unidades. Sin embargo, esto es tenido en cuenta por el equipo en el momento de realizar la comparación final.

4.1.5.2.2 Comunicación con el equipo

Ambas placas Arduino se deben comunicar con el equipo para intercambiar la información necesaria para establecer una comparación entre el resultado que arrojan los pines del Arduino monitorizado y el resultado que deberían arrojar, proporcionados por una traza que se recibe del Arduino monitorizado y que se genera gracias a una librería diseñada especialmente para ello.

Por lo tanto, existen dos vías de comunicaciones:

- **Arduino monitorizado con equipo:** a través del uso en el *sketch* de la librería Testduo proporcionada. Se generan unas trazas indicadoras de los pines a monitorizar y su resultado esperado, entre otra información. Estas trazas se envían al equipo utilizando una función específica de la librería, esta función, además, bloquea la ejecución del resto del *sketch* hasta que se haya verificado el resultado por parte del equipo. Si el resultado es correcto, el *sketch* continuará su ejecución, en caso contrario, el equipo cortará la conexión e informará la existencia del error.

- **Arduino monitor con equipo:** el equipo informa de los pines a monitorizar para que el Arduino monitor realice la lectura y envíe el resultado obtenido, con la ayuda de un *sketch* monitor.

Para comprender mejor el funcionamiento completo de un ciclo de comprobación de Testduo, se adjunta el siguiente diagrama:

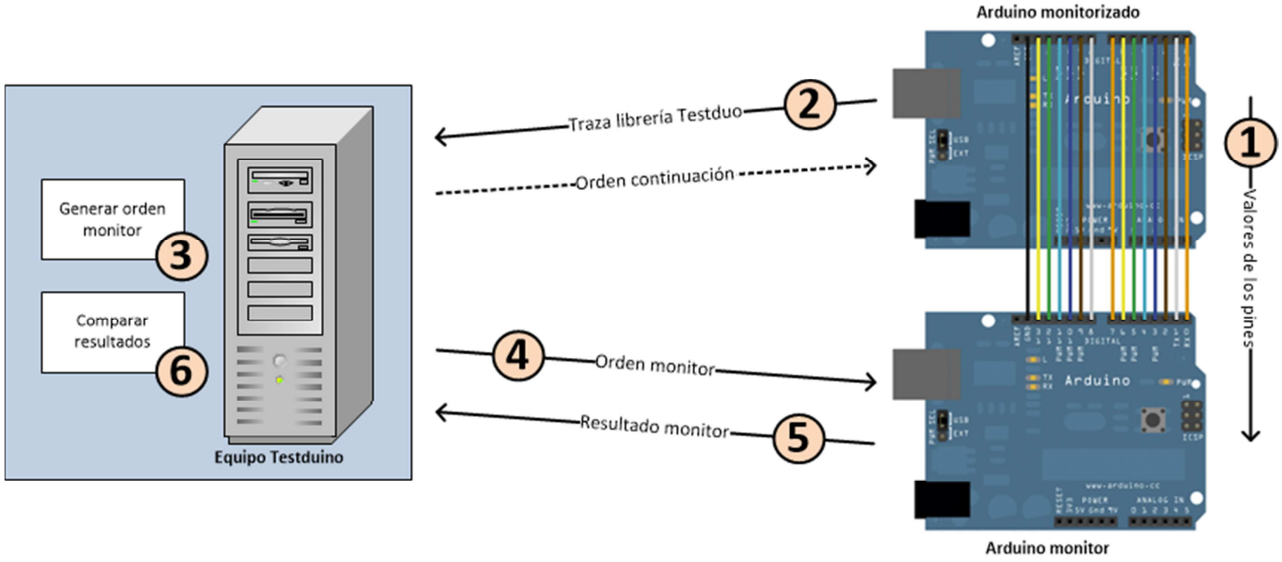


Ilustración 28: Protocolo de la comunicación Testduo

A continuación, se detallan cada una de las fases del diagrama anterior:

1. **Valores de los pines:** antes de realizar cualquier operación, el *sketch* del Arduino monitorizado establecerá los pines, cuyo valor estará disponible para su lectura gracias a la conexión física entre dicho Arduino y su monitor.

2. **Traza librería Testduo:** a decisión del desarrollador, éste instanciará un objeto de la librería Testduo, que configurará con información sobre los pines a monitorizar, su resultado esperado y una descripción de la traza opcional. A la hora de realizar la monitorización, después de establecerse los valores de los pines, se ejecutará un método del objeto instanciado que, basándose en la configuración introducida, generará una traza que se enviará al equipo, con la información descrita. Al enviarse la traza, la ejecución del *sketch* se congelará hasta que finalice el proceso de comprobación completo. La traza de Testduo sigue el formato siguiente:

   TESTDUOCHECKOUT.id.descripcion.numpin1#valuepin1.[…].numpin#valuepinN\n

   La traza es una cadena de caracteres que comienza con la palabra TESTDUOCHECKOUT, que identifica a cualquier traza Testduo del resto de contenido enviado por el puerto serie por el *sketch*. Se adjunta un identificador de la traza y una descripción, y posteriormente, separados por puntos, cada número de pin a monitorizar con su resultado esperado separado por una almohadilla. La cadena finaliza con un retorno de carro.

3. **Generar orden monitor:** una vez recibida la traza en el equipo, se activa un proceso de generación de la orden para el monitor, según los datos recibidos. La orden se genera según los números de pines recibidos en la traza del Arduino monitorizado. Esta orden es una cadena de caracteres, que se envía al Arduino monitor por su puerto serie, y que incluye los números de los pines a monitorizar.

4. **Orden monitor:** primeramente, se resetea el estado del Arduino monitor, reiniciando la ejecución de su *sketch*. Acto seguido, a través del puerto serie se realiza el envío de la orden generada. Su formato es el siguiente:

   numpin1.numpin2.numpin3.[…].numpinN\n

   El patrón es sencillo, son los números de pin separados por puntos. La cadena finaliza con un retorno de carro.

5. **Resultado monitor:** el Arduino monitor responde inmediatamente al equipo el valor leído de cada pin, en la misma posición que se envió su correspondiente número de pin en la orden, suplantándolo. El formato del resultado es el siguiente:

   valuepin1.valuepin2.valuepin3.[…].valuepinN\n

6. **Comparar resultados:** Testduino se encarga de realizar una comparación exacta de los valores digitales, y una comprobación aproximada de los valores de PWM, con una variación máxima de ±3 unidades. Si la comprobación concluye una coincidencia de todos los valores, Testduino comunicará al Arduino monitorizado que puede retomar su ejecución normal con una orden de continuación, enviándole por el puerto serie un carácter de retorno de carro '\n'. Por el contrario, si la comprobación encuentra una sola diferencia, el test se dará por fallido, informando el error por consola, y se finalizará la monitorización del *sketch*.

4.1.5.2.3 Librería Testduo

La librería Testduo es una librería desarrollada para Arduino compatible con cualquier *sketch*. Su objetivo es organizar la información sobre las trazas Testduo, generarlas, enviarlas y gestionar la parada y restablecimiento de la ejecución del *sketch*. Está implementada en C++ siguiendo las recomendaciones oficiales de Arduino y utilizando su API oficial.

La librería se compila, junto con el *sketch* y demás librerías utilizadas, y se carga en Arduino, ejecutándose en la placa hardware, y por lo tanto, no guardando más relación con Testduo que la comunicación serie necesaria, por lo que su funcionamiento no depende del equipo.

A continuación, se muestra el diagrama de clases de la librería, que se compone de sólo una clase, con un *sketch* de ejemplo para observar la/s relación/es de asociación posibles:

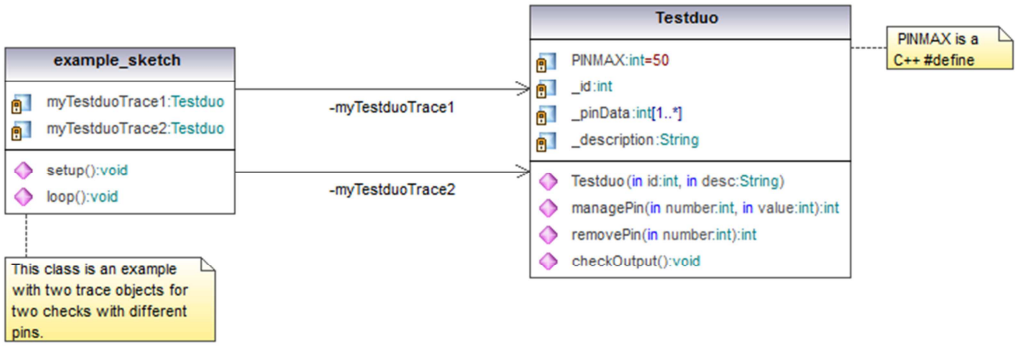


Ilustración 29: Diagrama de clases de la librería Testduo

Como se ha descrito, la clase *example_sketch* representa un *sketch* de prueba típico que, en este ejemplo, utiliza dos instancias de la clase Testduo de la librería para realizar dos comprobaciones, cada una con sus pines y valores esperados respectivos. A partir de este punto, se detalla la clase Testduo:

Identificador	Software empotrado Testduo
Nombre	Testduo
Tipo subcomponente	Clase (librería tipo Arduino)
Función	Proporcionar soporte al modo de testeo Testduo, soportando las estructuras de datos y comunicaciones necesarias. Cada objeto instanciado representa una traza con valores de pines diferentes en momentos diferentes.
Dependencias	Ninguna
Atributos	• **[#define] int PINMAX (=50):** rango de los números de pines soportados, por defecto entre 0-49 (este valor soporta todas las placas Arduino actuales a la venta). • **int _id:** identificador de traza. • **int _pinData [PINMAX]:** valores de cada uno de los pines, ordenados por su número de pin coincidiendo con su posición en este *array*. • **String _description:** descripción de la traza.
Métodos	• **managePin(int number, int value):** añade o modifica un valor esperado para un pin en concreto. • **removePin(int number):** elimina un valor esperado para un pin en concreto. Dicho pin no se monitorizará a la hora de realizar la siguiente comprobación de ésta traza. • **checkOutput():** lanza una traza de Testduo al equipo para iniciar la comprobación de los pines y bloquea la ejecución del *sketch* mientras dure el procedimiento de comprobación.

Tabla 127: Clase Testduo

4.2 Software

En este apartado, se describe en profundidad la arquitectura de la solución y su relación con el hardware, así como el programa de adaptación Testduino dedicado a realizar los test sobre las placas Arduino.

4.2.1 Modelo arquitectónico

El modelo arquitectónico establece cómo es la organización fundamental encarnada en sus componentes, las relaciones entre ellos y con el entorno, y los conceptos básicos que orientan su diseño y evolución.

Si tenemos en cuenta usuarios, procesos y flujos de datos y comunicaciones, se presenta la siguiente estructura:

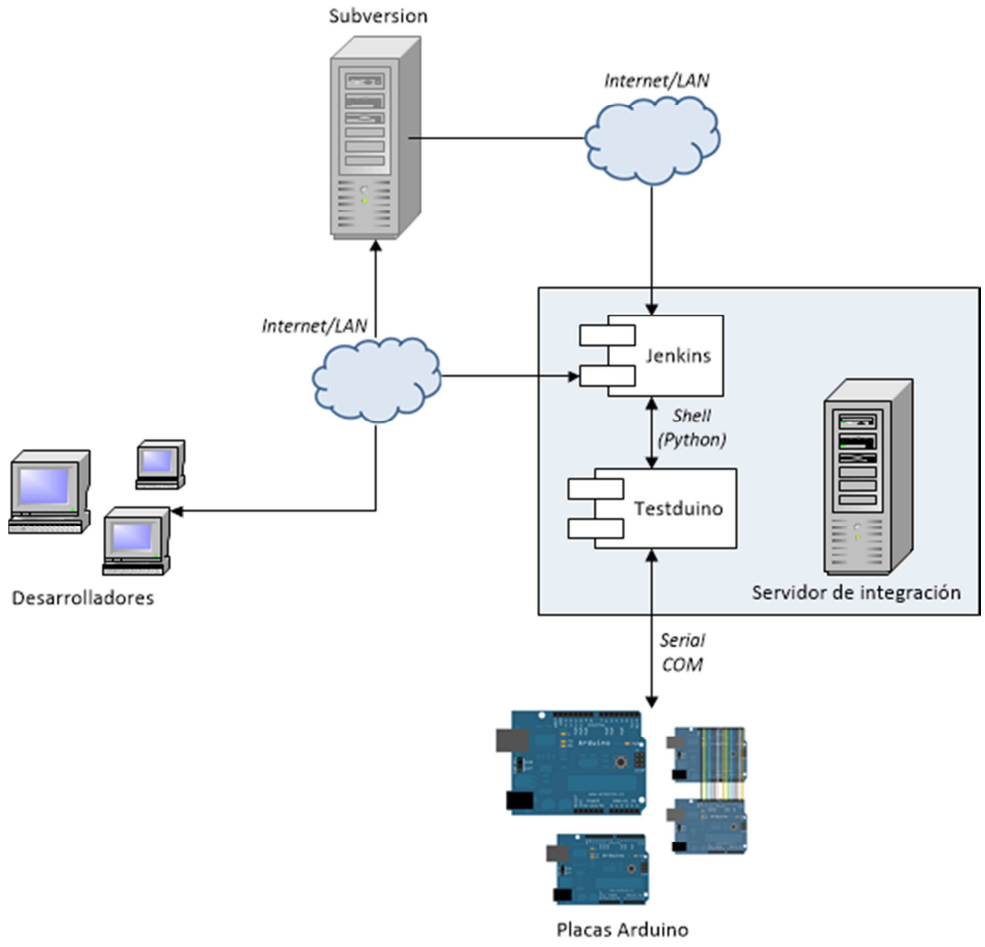


Ilustración 30: Estructura de la solución

Como se puede observar, la estructura de la solución sigue un modelo distribuido, donde cada parte de ella se puede ejecutar remotamente a través de red. Jenkins y Testduino comparte un mismo equipo, y su comunicación se basa en comandos y retornos de la consola de MS-DOS o Windows PowerShell. Por otra parte, la comunicación entre Testduino y las placas hardware es equivalente a la forma implementada en el IDE oficial de Arduino, es decir, a través del puerto serie.

El flujo de datos es reconocible en la estructura, comenzando por la actualización de las fuentes en el sistema Subversion por parte de los desarrolladores, y finalizando por el acceso a Jenkins por parte de los desarrolladores, o informe de los resultados de Jenkins a los mismos.

4.2.1.1 Arquitectura general

La arquitectura general de la solución es orientada a eventos, debido a que el sistema sólo comienza a realizar acciones al producirse un evento, mientras tanto el sistema sólo se comporta como un servicio pasivo[55]. Este esquema se reproduce en el programa de adaptación Testduino, en el cual, principalmente su actitud es pasiva hasta que las placas Arduino envían información sobre su ejecución.

Desde el punto de vista del sistema de integración continua, Jenkins es un programador de tareas, que se suceden a partir de eventos. Esta clase de eventos son: coincidencia con una programación, actualización en el Subversion u orden de ejecución inmediata. Además, se deben incorporar eventos relacionados con el testeo: informe de éxito o informe de error.

El programa de adaptación Testduino varía su flujo de ejecución según los eventos que ocurriesen en las placas Arduino en pruebas. Los eventos posibles son: finalización exitosa y error localizado. El modo de monitorización Testduo incorporaría eventos tales como: traza de Testduo y respuesta del Arduino monitor.

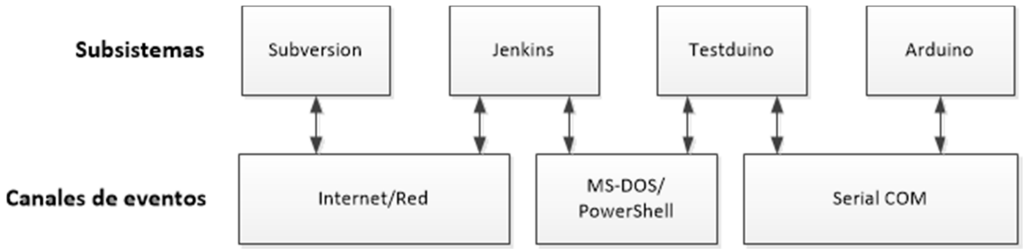


Ilustración 31: Modelo arquitectónico de eventos general

Según lo expuesto, activadores o generadores de eventos son: el hardware Arduino, el sistema de control de versiones Subversion y el propio Jenkins para su programación y generación de informes. Como motores de procesamiento de eventos podemos contemplar al programa de adaptación de Testduino, soportando eventos de Jenkins y del hardware Arduino, y el sistema de integración continua Jenkins. Los canales de eventos son los ya mencionados: conexiones TCP/IP de internet o red privada, comandos/retornos de consola y comunicaciones serie. En todo caso, esta explicación se refiere a una solución básica sin complementos extra en cualquiera de los componentes, que bien podrían variar los generadores de eventos.

Por otra parte, cada componente o subsistema de la solución posee su arquitectura propia, desde su punto de vista como un sistema independiente que se relaciona con sistemas externos. Por lo tanto, aunque desde la perspectiva de la solución el sistema completo funcione bajo respuesta de eventos, cada componente, desde su propia perspectiva, posee su propia implementación basada en su diseño por terceros, salvo en el caso del programa de adaptación Testduino, cuyo modelo se explica más adelante.

4.2.1.2 Subversion y Jenkins

Subversion y Jenkins son sistemas orientados a red, que son autónomos en sí mismos y no dependen de ningún otro sistema, sólo completan la solución Testduino.

Subversion es un sistema de control versiones basado en el modelo cliente-servidor, y por lo tanto, su modelo arquitectónico es el estándar de este tipo de arquitecturas: Subversion actúa como servidor central de repositorios y los clientes, en este caso Jenkins y los desarrolladores, le realizan peticiones de diversos tipos, como actualización o descarga, que provocan transmisiones del servidor al cliente o viceversa, en las cuales, Subversion siempre actúa como servidor conectándose el cliente a él.

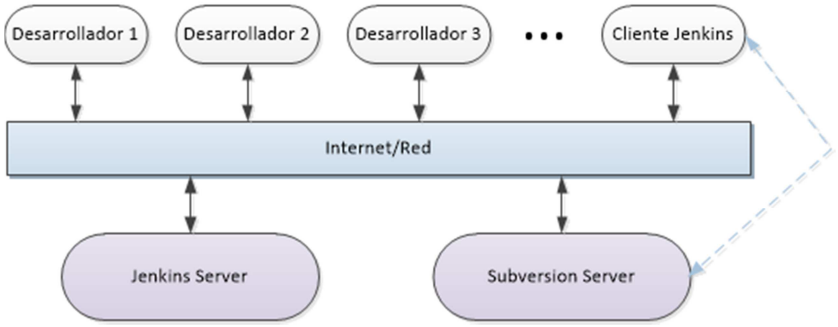


Ilustración 32: Modelo arquitectónico cliente-servidor

Jenkins se aloja en un servidor Apache Tomcat, y aunque también puede funcionar como un servicio de Windows, la plataforma sobre la que se ubica es la misma. Al tratarse de un servidor, en principio hablamos de una arquitectura cliente-servidor estándar, que si bien, adjuntando el servidor web, acaba derivando en una arquitectura modelo vista controlador (MVC), cuyos componentes se organizan de la siguiente forma:

- **Vista:** servidor web dedicado a mostrar las páginas webs generadas de Jenkins.
- **Controlador:** dedicado a la recepción de eventos y ejecución de acciones.
- **Modelo:** estructuras de datos y servicios ofrecidos al usuario.

Además, Jenkins incorpora diversos servicios web para control remoto y monitorización externa, lo que le atañe características de una arquitectura de servicios web.

Las conexiones entre Subversion y Jenkins son a través de red mediante el protocolo SVN exclusivo de Subversion. Este protocolo es actúa como modelo cliente-servidor, y por lo tanto, Jenkins incorpora un cliente propio de Subversion para las comunicaciones. El protocolo SVN es una capa propia que actúa sobre el protocolo TCP en el puerto 3690.

4.2.1.3 Programa de adaptación Testduino

La organización del programa de adaptación está orientada a través de una arquitectura que basa el flujo de ejecución en eventos ocurridos, que son transmitidos al programa, encontrándose éste siempre en un estado pasivo de escucha. Esta arquitectura se replica con cada monitorización, ya que cada evento recibido lo gestiona el monitor específico para la placa que lo envía. Por lo tanto, el diseño arquitectónico establecido se encuentra basado en eventos de procesos paralelos, es decir, es una arquitectura propia que mezcla los conceptos de orientación a eventos y la arquitectura de procesos paralelos.

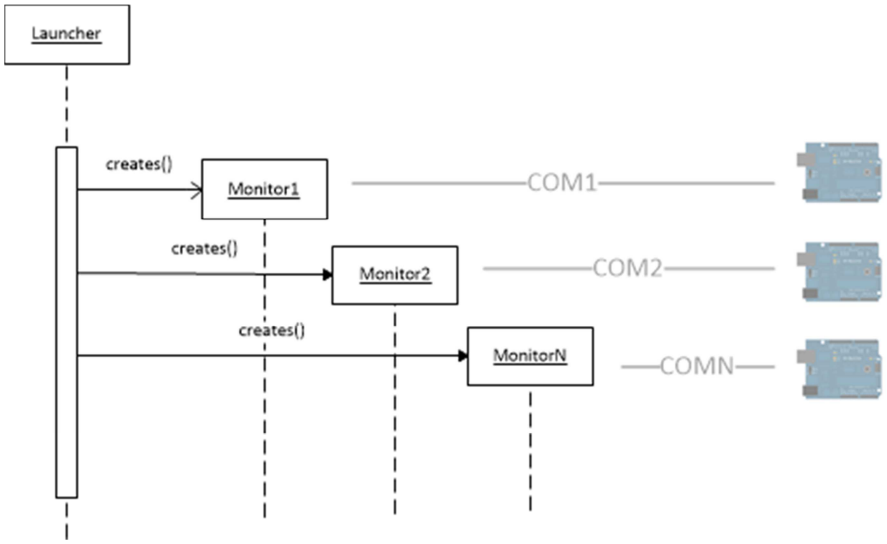


Ilustración 33: Modelo arquitectónico de procesos paralelos y eventos

Como se puede observar, el proceso *Launcher*, encargado de compilar y cargar los diferentes *sketches*, será el que inicie cada uno de los procesos monitores paralelos que controlarán eventos que ocurran por su correspondiente puerto serie asignado.

Los hilos son creados mediante la API de alto nivel de Python proporcionada por la clase *Thread*, que aporta lo justo y necesario para la consecución de la arquitectura diseñada. Python posee otra API de más bajo nivel, y que también se puede utilizar directamente, que es utilizada por la API de alto nivel para comunicarse con la API del sistema operativo.

4.2.2 Plataformas

Las plataformas son una parte importante de la arquitectura para el correcto funcionamiento de la solución, ya que, son los cimientos del software principal que se ejecuta en los últimos niveles. Es necesaria la existencia de una perfecta integración a todos los niveles, desde el sistema operativo hasta la capa de software final.

Subversion funciona sobre su propia plataforma y servidor, aunque también puede acoplarse a servicios externos en la nube como Dropbox, pero no es una característica que influya en la solución o esté controlada por ésta. En el polo opuesto, el programa de adaptación de Testduino y su integración con Jenkins sí dependen de una variedad de plataformas, soportes y librerías que son importantes y afectan al funcionamiento, además de ser necesarias.

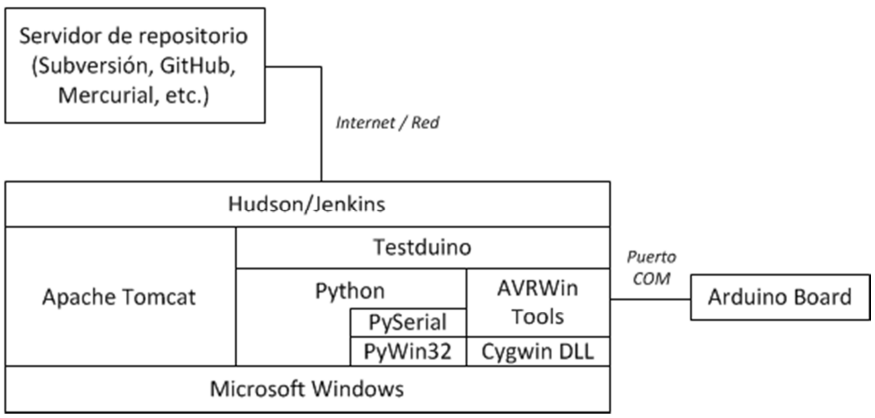


Ilustración 34: Esquema de plataformas y librerías

A continuación, se procede al detalle de cada plataforma/librería, su relación con el resto y su función:

- **Microsoft Windows:** es la base de cualquier software, el sistema operativo. Establecerá las comunicaciones con el hardware Arduino a través de los correspondientes controladores, y controlará el sistema de ficheros utilizado por la solución.

- **Apache Tomcat:** es un contenedor de *servlets*. Tanto Hudson como su clon Jenkins son un *servlet* que es necesario instalar en un contenedor de este tipo. Jenkins soporta una instalación como un servicio de Windows que prescinde de realizar una instalación típica de Apache Tomcat, aunque en realidad, realiza una instalación automatizada oculta junto con su *servlet* WAR.

- **Python:** el programa de adaptación Testduino se encuentra desarrollado en este lenguaje, por lo que es necesaria la instalación de un intérprete de Python 3 de 32 bits para su funcionamiento. Por otra parte, las funcionalidades de Testduino dependen de dos librerías que no vienen de serie en el intérprete:

    - **PySerial:** librería realizada en Python que permite establecer una comunicación, en ambos sentidos, por cualquier puerto serie del sistema.

    - **PyWin32:** librería que permite al intérprete utilizar la API de Win32 para interactuar de forma efectiva con los sistemas Microsoft Windows. En este caso, PySerial requiere de ésta librería para accionar las funciones necesarias del API de Windows que controlan los puertos de comunicación.

- **AVRWin Tools:** utilidades de compilación y carga para microcontroladores AVR, adaptadas a sistemas Microsoft Windows. Testduino requiere de las utilidades incorporadas en el IDE oficial de Arduino, así como de su configuración.

    - **Cygwin DLL:** librería que permite utilizar adaptaciones de aplicaciones desarrolladas en un principio para sistemas tipo UNIX. Necesaria para la ejecución de las herramientas AVRWin. Esta librería está incorporada en el IDE oficial de Arduino necesario.

En el esquema anterior se puede observar que Jenkins es la plataforma final que el usuario verá y podrá interactuar, y Testduino es el programa de adaptación que se ejecuta en una capa inferior y sólo envía resultados a Jenkins.

4.2.3 Diseño detallado de componentes

Esta solución se basa en la integración de varias piezas de software de terceros con la ayuda de una adaptación realizada especialmente para ello: Testduino.

Testduino debe localizar los *sketches* de prueba en su última versión y ejecutarlos sobre las placas Arduino, estando pendiente del resultado final. Testduino se ayuda de Jenkins para realizar la integración continua, descargando éste las últimas versiones e informando al usuario de los resultados de una forma amigable. Al ser Jenkins y Subversion software de terceros, no serán analizados sus diseños internos, dedicando este apartado al software que ha sido necesario desarrollar para lograr el funcionamiento con Arduino de estas plataformas.

Seguidamente, se muestra el diagrama de componentes según las directrices del lenguaje gráfico UML.

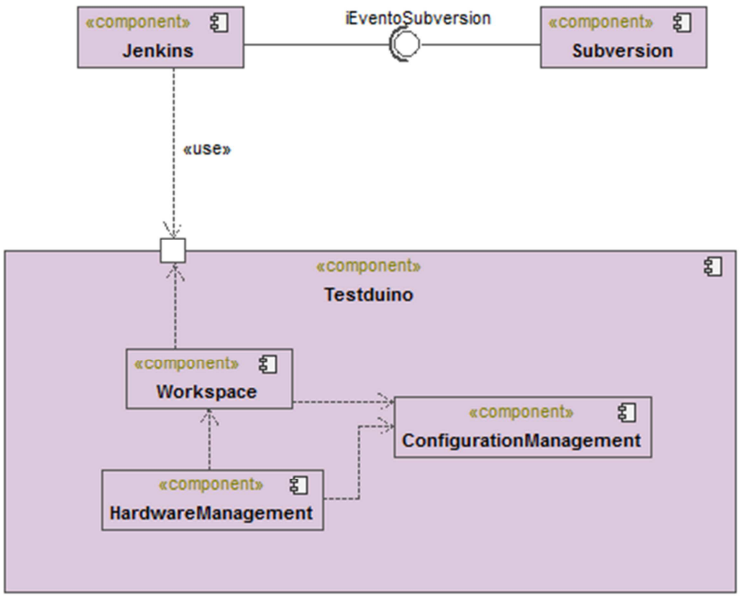


Ilustración 35: Diagrama de componentes

La solución se encuentra compuesta de tres componentes principales:

- **Subversion:** es el sistema de control de versiones utilizado, que actúa como repositorio de pruebas y librerías. Se encarga de gestionar los desarrollos y sus ramas, suministrando versiones y actualizando cambios gestionando su control.

- **Jenkins:** es el sistema de integración continua, que aplica los conceptos de la integración con el repositorio de versiones y el gestor de pruebas que, en este caso, es Testduino. Se encarga de verificar que se hayan realizado cambios y de proporcionarlos a Testduino para su prueba. Además, cumple la función de GUI mostrando un sitio web de gestión de la integración al usuario.

- **Testduino:** es el programa de adaptación que permite que Jenkins ejecute pruebas unitarias sobre hardware Arduino. Sus funcionalidades son:

    - **Gestionar el espacio de trabajo** administrando la **configuración** del hardware y de monitorización de los *sketches*.

    - **Compilar** los *sketches*.

- **Cargar** los *sketches* en las placas.
- **Monitorizar** los procesos de **compilación**, **carga** y **ejecución**.
- **Gestionar la comunicación** en el modo **Testduo** para permitir la monitorización de una placa Arduino a otra.
- **Almacenar los errores** encontrados y **notificarlos** al usuario.

4.2.3.1 Especificación de componentes

A continuación, se explican exhaustivamente todos los componentes que conforman la solución, utilizando, para ello, la siguiente plantilla:

Identificador:	Componente X
Nombre	
Tipo	
Función	
Interfaces	
Dependencias	
Proceso	
Datos	
Recursos	
Origen	

Tabla 128: Plantilla de componentes de la solución

- **Identificador:** es el código único que identifica a cada componente. Su forma es "Componente" más un número.
- **Nombre:** nombre con el que se representa el componente.
- **Tipo:** señala el tipo de componente, en el caso de que el componente forme parte de un grupo de componentes.
- **Función:** explica el objeto del componente en la solución.
- **Interfaces:** interfaces que tiene que implementar o requiere en su desarrollo.
- **Dependencias**: se enumeran los componentes que dependen de este componente para su desarrollo, pero sin necesidad de ceñirse a una determinada interfaz.
- **Proceso:** describe el funcionamiento y el comportamiento en el momento que el componente participe en la ejecución.
- **Datos:** señala los datos que utiliza el componente.
- **Recursos:** señala los recursos que tiene disponible el componente para realizar su función.
- **Origen**: indica los requisitos software de los que procede el diseño del componente.

4.2.3.1.1 Sistemas de terceros

Identificador:	Componente 1
Nombre	Subversion
Tipo	No se aplica
Función	Gestionar las versiones de las librerías y *sketches* a testear.
Interfaces	iEventoSubversion (implementa)
Dependencias	Ninguna
Proceso	1. El componente se encuentra pasivo esperando eventos. 2. Si ocurre una actualización, se deja constancia de ello en el repositorio, a través de su *log* de eventos. 3. Responde a las peticiones de Jenkins o de los desarrolladores, si las hubiera, a través de su interfaz.
Datos	• Recibe cambios en las librerías (opcional) y/o los *sketches*. • Envía la versión de librerías (opcional) y/o los *sketches* solicitada.
Recursos	• Versiones de librerías Arduino para pruebas (opcional). • Versiones *sketches* de pruebas. • *Log* de eventos. • Configuración propia.
Origen	F-SR-01 y NF-IC-10.

Tabla 129: Componente 1

Identificador:	Componente 2
Nombre	Jenkins
Tipo	No se aplica
Función	Monitorizar el repositorio de versiones actualizando su versión local con la última versión, ejecutar Testduino según los eventos programados y proporcionar una interfaz de usuario para gestión y resultados.
Interfaces	iEventoSubversion (requiere)
Dependencias	Testduino
Proceso	1. El componente se encuentra pasivo esperando eventos. Dos posibilidades según configuración: A. Cuando ocurre un evento de Subversion, se conecta la interfaz de éste para recibir los últimos datos y ejecutar Testduino. B. Cuando la programación genera un evento, se conecta a la interfaz de Subversion para recibir los últimos datos y ejecutar Testduino.
Datos	• Recibe la última versión de las librerías (opcional) y/o *sketches* de prueba.
Recursos	• Versión local de librerías Arduino para pruebas (opcional). • Versión local de *sketches* de pruebas. • Configuración propia. • *Logs* de ciclos de testeo.
Origen	F-SR-01, F-SR-04, F-SR-05, F-SR-06, F-SR-07, F-SR-08, F-SR-09, F-SR-10, F-SR-11, F-SR-12, F-SR-18, F-SR-19, NF-IC-08, NF-IC-09, NF-IC-10, NF-IS-05, NF-IS-06, NF-IU-01 y NF-IU-02.

Tabla 130: Componente 2

4.2.3.1.2 Testduino

Identificador:	Componente 3
Nombre	Testduino
Tipo	No se aplica
Función	Compilar y cargar los *sketches* de prueba, monitorizando su proceso y ejecución en el hardware Arduino e informar de los resultados, y establecer los protocolos de comunicación para la monitorización Testduo entre placas Arduino.
Interfaces	No requiere
Dependencias	Ninguna
Proceso	1. El componente comienza su ejecución por mandato de Jenkins. 2. Se carga y se comprueba la configuración general de hardware, rutas de acceso y de puertos. 3. Se lee cada *sketch* comprobando el tipo de pruebas: normal, en puerto específico, Testduo o no probar, generando el *workspace*. 4. Se cargan y monitorizan los *sketches* de modo Testduo (procedimiento de comunicación explicado en el apartado Hardware). 5. Se cargan y monitorizan los *sketches* de modo puerto específico. 6. Se cargan y monitorizan los *sketches* de modo normal. 7. Se recopilan resultados y se retornan a Jenkins.
Datos	• Envía el resultado de la ejecución.
Recursos	• Versión local de librerías Arduino para pruebas (opcional). • Versión local de *sketches* de pruebas. • Configuración de hardware y rutas de acceso. • Configuración de puertos serie.
Origen	F-SR-03, F-SR-12, F-SR-13, F-SR-14, F-SR-15, F-SR-16, F-SR-17, F-SR-19, F-SR-20, F-SR-24, F-SR-28, F-SR-29, F-SR-30, F-SR-31, F-SR-33, NF-IC-01, NF-IC-02, NF-IC-03, NF-IC-04, NF-IC-05, NF-IC-06, NF-IC-07, NF-IC-09, NF-IH-01, NF-IH-02, NF-IH-03, NF-IH-04, NF-IS-01, NF-IS-02, NF-IS-03, NF-IS-04, NF-IS-11, NF-IS-12, NF-IU-03, NF-PO-01, NF-PO-02, NF-TM-01, NF-TM-02 y NF-TM-03.

Tabla 131: Componente 3

Identificador:	Componente 4
Nombre	Workspace
Tipo	Testduino
Función	Gestiona los parámetros de modo especiales incluidos en los *sketches*. Estos parámetros especifican características de la prueba a utilizar y son incluidos como comentarios en el código fuente de los *sketches*. Además, gestiona la localización de los *sketches* y el directorio temporal de manejo interno de Testduino.
Interfaces	No requiere
Dependencias	ConfigurationManagement
Proceso	1. Recibe la ruta del espacio de trabajo (*workspace*) y busca todos los *sketches* recursivamente en su estructura de directorios. 2. Copia la estructura a un directorio interno de trabajo de Testduino. 3. Por cada *sketch*, localiza en los comentarios del código fuente si se desea un modo de prueba normal, Testduo, puerto específico o no probar.
Datos	• Recibe las rutas de acceso de los *sketches* de prueba. • Envía las rutas de acceso al directorio interno donde se copian los *sketches* de prueba.
Recursos	• Versión local de librerías Arduino para pruebas (opcional). • Versión local de *sketches* de pruebas. • Directorio interno.
Origen	F-SR-02, F-SR-21, F-SR-22, F-SR-25, F-SR-26, F-SR-27, NF-IH-03, NF-IH-04, NF-SE-01 y NF-SE-02.

Tabla 132: Componente 4

Identificador:	Componente 5
Nombre	HardwareManagement
Tipo	Testduino
Función	Compilar los *sketches*, cargarlos en las placas Arduino y monitorizar su ejecución, a través de trazas por el puerto serie. En el modo Testduo, debe controlar el protocolo de trazas para la monitorización entre placas.
Interfaces	No requiere
Dependencias	ConfigurationManagement y Workspace
Proceso	1. Establece los comandos de compilación y carga según la configuración hardware de la placa y la configuración de modo de prueba del *sketch*. 2. Ejecuta los comandos de compilación y carga. 3. Inicia una comunicación con la placa Arduino específica a través del puerto serie. Si estamos en el modo Testduo: a) Carga en el Arduino monitor el *sketch* de monitorización de pines. b) Cuando recibe una traza Testduo, informa de los pines al Arduino monitor. c) Recibe los valores del Arduino monitor y comprueba la coincidencia con los esperados especificados en la traza Testduo. d) En caso positivo, ordena continuar la ejecución del Arduino monitorizado. En caso de no coincidencia, cierra las comunicaciones con los Arduino y se reporta error. 4. Cuando ocurre un error o se finaliza la ejecución, cierra la comunicación informando del resultado.
Datos	• Recibe las rutas de las librerías, Arduino IDE y *sketches* de prueba. • Recibe las trazas de monitorización normal o Testduo. • Envía información sobre pines monitorizados (sólo Testduo). • Recibe valores de los pines monitorizados (sólo Testduo). • Envía resultado de la prueba.
Recursos	• Versión interna de librerías Arduino para pruebas (opcional). • *Sketch* de prueba. • Ficheros objeto y binarios del *sketch* de prueba. • Herramientas de compilación y carga y librerías básicas (Arduino IDE).
Origen	F-SR-13, F-SR-14, F-SR-15, F-SR-16, F-SR-20, F-SR-23, F-SR-25, F-SR-30, F-SR-34, F-SR-35, F-SR-36, NF-IC-03, NF-IC-05, NF-IS-08, NF-IS-09, NF-IS-10 y NF-IS-12.

Tabla 133: Componente 5

Identificador:	Componente 6
Nombre	ConfigurationManagement
Tipo	Testduino
Función	Establecer la configuración de las placas Arduino, puertos serie disponibles con hardware asociado, parámetros de monitorización generales y rutas de acceso a los *sketches*, librerías y Arduino IDE.
Interfaces	No requiere
Dependencias	Ninguna
Proceso	1. Carga la configuración de rutas de acceso y hardware general. 2. Carga la configuración de puertos serie y hardware asociado. 3. Atiende peticiones de información del resto de componentes de Testduino.
Datos	• Recibe datos de configuración. • Envía datos de configuración solicitados.
Recursos	• Fichero de configuración de Testduino (rutas y parámetros generales). • Fichero de configuración de hardware asociado y puertos serie.
Origen	F-SR-03, F-SR-13, F-SR-27, F-SR-32, F-SR-37, F-SR-38, NF-IC-02, NF-IS-07, NF-TM-01, NF-TM-02 y NF-TM-03.

Tabla 134: Componente 6

4.2.3.2 Diagrama de clases de Testduino

En este apartado, se muestran las clases o subcomponentes resultantes que componen la implementación del programa de adaptación de Testduino, según la normativa de modelado UML, en la página siguiente.

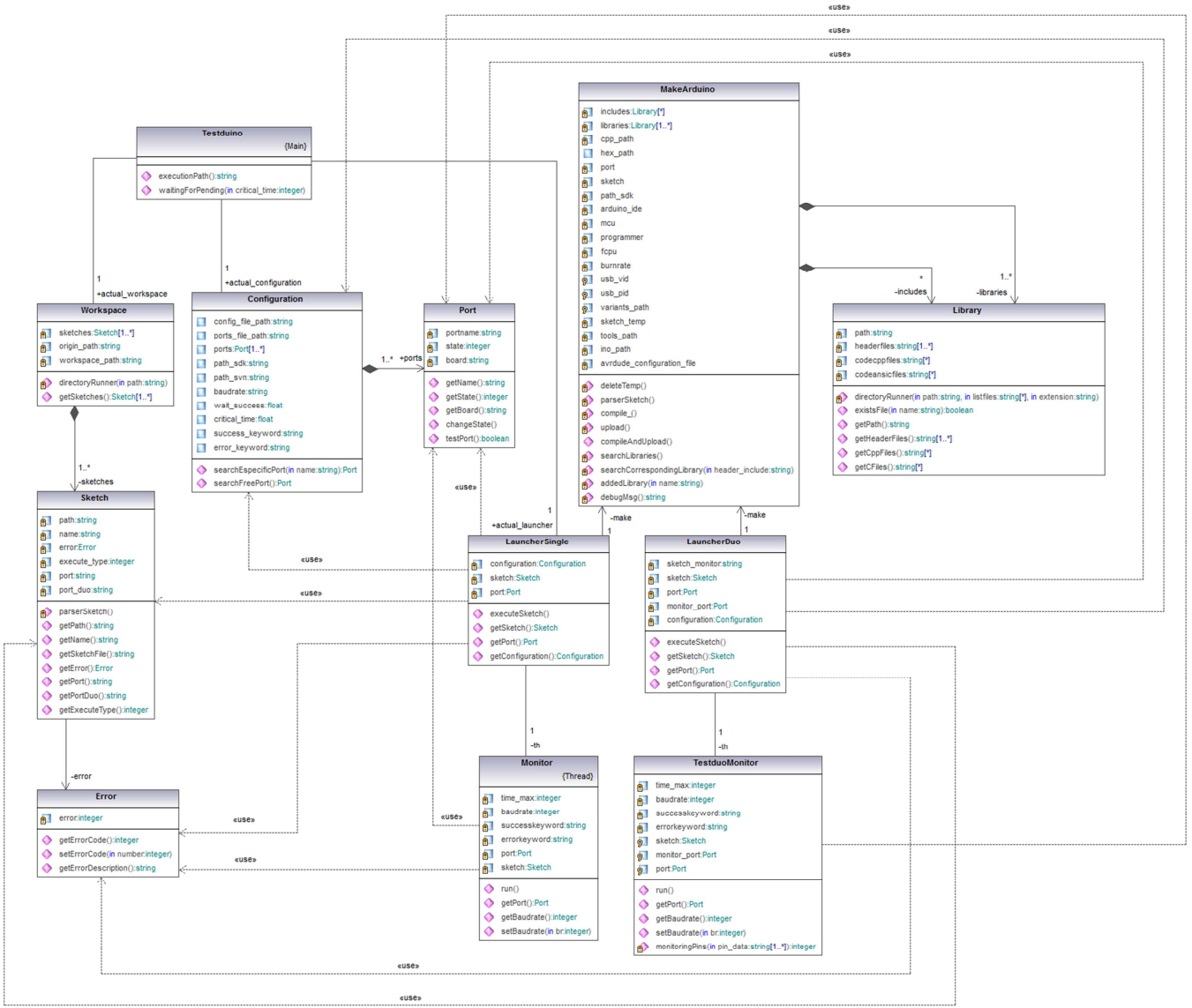


Ilustración 36: Diagrama de clases

4.2.3.3 Especificación de subcomponentes

En este apartado, se detalla el diseño de clases realizado para el programa de adaptación Testduino. Cada clase o subcomponente pertenece a un componente de Testduino o al propio componente Testduino principal, según la temática de su funcionalidad.

El formato utilizado para la descripción es el siguiente:

Identificador:	Subcomponente X
Nombre	
Componente	
Tipo subcomponente	
Función	
Dependencias	
Atributos	
Métodos	

Tabla 135: Plantilla de subcomponentes de Testduino

- **Identificador:** es el código único que identifica a cada subcomponente. Su formato es "Subcomponente X", siendo X un número identificativo.
- **Nombre:** denominación para el subcomponente.
- **Componente:** componente más inmediato al que pertenece.
- **Tipo subcomponente:** indica el tipo de subcomponente, pudiendo ser clase, fichero o interfaz, entre otros.
- **Función:** describe la funcionalidad del subcomponente.
- **Dependencias:** enumera los subcomponentes necesarios para el funcionamiento de este subcomponente, pero sin necesidad de ceñirse a una determinada interfaz.
- **Atributos:** enumera y detalla los atributos del subcomponente (clase).
- **Métodos:** enumera y describe los métodos del subcomponente (clase o interfaz).

Identificador:	Subcomponente 1
Nombre	Testduino
Componente	Testduino
Tipo subcomponente	Clase
Función	Es la clase principal de ejecución. Sus tareas son administrar el espacio de trabajo (*workspace*) y la batería de pruebas gestionando los puertos serie libres y los *sketches* que se lanzan a través de ellos. Por último, retorna el resultado de la ejecución.
Dependencias	Workspace, Configuration, LauncherSingle y LauncherDuo
Atributos	No se aplica
Métodos	• ***Main***: ejecución principal que genera el *workspace* y ordena el lanzamiento de pruebas a través de los puertos libres. • **executionPath():** devuelve la ruta absoluta donde se está ejecutando Testduino. • **waitingForPending():** espera hasta que terminen todas las monitorizaciones.

Tabla 136: Subcomponente 1

Identificador:	Subcomponente 2
Nombre	Workspace
Componente	Workspace
Tipo subcomponente	Clase
Función	Representar el espacio de trabajo, generando el directorio interno, buscando y administrando los *sketches* encontrados.
Dependencias	Sketch
Atributos	• **sketches[]:** lista de *sketches* encontrados. • **origin_path**: ruta absoluta original configurada donde se encuentran los *sketches*. • **workspace_path:** ruta absoluta del directorio interno usado por Testduino para manejo de *sketches*.
Métodos	• **directoryRunner(path):** recorre una ruta de forma recursiva buscando y almacenando todos los *sketches* encontrados. • **getSketches():** devuelve la lista de *sketches* encontrados.

Tabla 137: Subcomponente 2

Identificador:	Subcomponente 3
Nombre	Sketch
Componente	Workspace
Tipo subcomponente	Clase
Función	Representar un *sketch* y sus atributos: nombre, rutas, puerto configurado y errores.
Dependencias	Error
Atributos	• **path:** ruta absoluta del *sketch*. • **name:** nombre del *sketch*. • error: objeto de la clase Error que almacena un error encontrado si lo hubiere. • **execute_type:** tipo de ejecución configurado (no probar, normal, puerto específico o Testduo). • **port:** puerto específico (sólo si se ha configurado). • **port_duo:** puerto para Testduo (sólo si se ha configurado).
Métodos	• **parserSketch():** busca trazas de Testduino con configuraciones sobre su tipo de ejecución. • **getPath():** devuelve la ruta absoluta del *sketch*. • **getName():** devuelve el nombre del *sketch*. • **getSketchFile():** devuelve el nombre de fichero del *sketch*. • **getError():** devuelve el objeto Error. • **getPort():** devuelve el puerto especificado (sólo si está configurado). • **getExecuteType():** devuelve el tipo de ejecución.

Tabla 138: Subcomponente 3

Identificador:	Subcomponente 4
Nombre	Error
Componente	Workspace
Tipo subcomponente	Clase
Función	Almacenar el error cuando se produzca. Los errores pueden ser de compilación, carga, ejecución o monitorización.
Dependencias	Ninguna
Atributos	• **error:** almacena el tipo de error producido.
Métodos	• **getErrorCode():** devuelve el tipo de error producido, si se ha producido alguno. • **setErrorCode():** establece un error. Función utilizada por los monitores y lanzadores. • **getErrorDescription():** devuelve un texto explicativo del tipo de error.

Tabla 139: Subcomponente 4

Identificador:	Subcomponente 5
Nombre	Configuration
Componente	ConfigurationManagement
Tipo subcomponente	Clase
Función	Almacenar la configuración hardware y general de Testduino.
Dependencias	Port
Atributos	• **config_file_path:** ruta absoluta del fichero de configuración principal. • **ports_file_path:** ruta absoluta del fichero de configuración de puertos y hardware. • **ports[]:** listado de puertos. • **path_sdk:** ruta absoluta del Arduino IDE. • **path_svn:** ruta absoluta de los *sketches* de prueba. • **baudrate:** tasa de baudios para la monitorización serie. • **wait_success:** tiempo de espera máximo para éxito. • **critical_time:** tiempo de espera máximo para finalización de pruebas en proceso de monitorización. • **success_keyword:** palabra de traza que simboliza éxito. • **error_keyword:** palabra de traza que simboliza error o fallo.
Métodos	• **searchEspecificPort(name):** devuelve el puerto especificado si se encuentra libre. • **searchFreePort():** busca y devuelve el primer puerto libre encontrado.

Tabla 140: Subcomponente 5

Identificador:	Subcomponente 6
Nombre	Port
Componente	ConfigurationManagement
Tipo subcomponente	Clase
Función	Representar puerto y almacenar su estado e información sobre el hardware conectado.
Dependencias	Ninguna
Atributos	• **portname:** denominación del puerto. • **state:** estado del puerto (libre u ocupado). • **board:** tipo de placa Arduino conectada al puerto.
Métodos	• **getName():** devuelve la denominación del puerto. • **getState():** devuelve el estado actual del puerto. • **getBoard():** devuelve el tipo de placa Arduino conectada. • **changeState():** alterna el estado del puerto. • **testPort():** prueba y devuelve si el puerto está disponible.

Tabla 141: Subcomponente 6

Identificador:	Subcomponente 7
Nombre	LauncherSingle
Componente	HardwareManagement
Tipo subcomponente	Clase
Función	Realiza el lanzamiento de pruebas en cualquier modo de ejecución salvo Testduo. Monitoriza el proceso de compilación de carga y crea el monitor para la ejecución.
Dependencias	Configuration, Sketch, Monitor, Port y Error
Atributos	• **configuration:** configuración del lanzamiento. • **sketch:** *sketch* del lanzamiento. • **port:** puerto para el lanzamiento.
Métodos	• **executeSketch():** compila y carga el *sketch* y genera el monitor de ejecución. También almacena los errores ocurridos. • **getSketch():** devuelve el *sketch* del lanzamiento. • **getPort():** devuelve el puerto para el lanzamiento. • **getConfiguration():** devuelve la configuración del lanzamiento.

Tabla 142: Subcomponente 7

Identificador:	Subcomponente 8
Nombre	Monitor
Componente	HardwareManagement
Tipo subcomponente	Clase
Función	Monitoriza la ejecución de un *sketch* sobre una placa Arduino en cualquier tipo de ejecución salvo Testduo. Esta clase hereda de la clase Thread de Python, ya que genera un hilo de ejecución.
Dependencias	Port, Sketch y Error
Atributos	• **time_max:** tiempo máximo de monitorización. • **baudrate:** tasa de baudios de la comunicación serie. • **successkeyword**: palabra clave de traza que significa éxito. • **errorkeyword:** palabra clave de traza que significa error o fallo. • **port:** puerto de monitorización. • **sketch:** *sketch* a monitorizar.
Métodos	• **run():** método de monitorización que se ejecuta sobre un hilo de ejecución. • **getPort():** devuelve el puerto de monitorización. • **getBaudrate():** devuelve la tasa de baudios. • **setBaudrate(br):** establece una nueva tasa de baudios.

Tabla 143: Subcomponente 8

Identificador:	Subcomponente 9
Nombre	LauncherDuo
Componente	HardwareManagement
Tipo subcomponente	Clase
Función	Realiza el lanzamiento de pruebas en modo de ejecución Testduo. Monitoriza el proceso de compilación de carga del *sketch* de prueba y el *sketch* de monitorización y crea el monitor para la ejecución.
Dependencias	Configuration, Sketch, Monitor, Port y Error
Atributos	• **configuration:** configuración del lanzamiento. • **sketch:** *sketch* del lanzamiento. • **sketch_monitor:** ruta del *sketch* de monitorización. • **port:** puerto para el lanzamiento. • **monitor_port:** puerto de la placa Arduino de monitorización.
Métodos	• **executeSketch():** compila y carga el *sketch* y el *sketch* de monitorización y genera el monitor de ejecución. También almacena los errores ocurridos. • **getSketch():** devuelve el *sketch* del lanzamiento. • **getPort():** devuelve el puerto para el lanzamiento. • **getConfiguration():** devuelve la configuración del lanzamiento.

Tabla 144: Subcomponente 9

Identificador:	Subcomponente 10
Nombre	TestduoMonitor
Componente	HardwareManagement
Tipo subcomponente	Clase
Función	Monitoriza la ejecución de un *sketch* sobre una placa Arduino en modo Testduo, es decir, monitorizando además los pines a través de una segunda placa Arduino. Esta clase hereda de la clase Thread de Python, ya que genera un hilo de ejecución.
Dependencias	Port, Sketch y Error
Atributos	• **time_max:** tiempo máximo de monitorización. • **baudrate:** tasa de baudios de la comunicación serie. • **successkeyword**: palabra clave de traza que significa éxito. • **errorkeyword:** palabra clave de traza que significa error o fallo. • **port:** puerto de monitorización. • **monitor_port:** puerto de la placa de monitorización. • **sketch:** *sketch* a monitorizar.
Métodos	• **run():** método de monitorización que se ejecuta sobre un hilo de ejecución. • **getPort():** devuelve el puerto de monitorización. • **getBaudrate():** devuelve la tasa de baudios. • **setBaudrate(br):** establece una nueva tasa de baudios. • **monitoringPins(pin_data):** realiza la monitorización de pines mediante el Arduino monitor gestionando la comunicación.

Tabla 145: Subcomponente 10

Identificador:	Subcomponente 11
Nombre	MakeArduino
Componente	HardwareManagement
Tipo subcomponente	Clase
Función	Realizar el proceso de compilación y carga del *sketch* en la placa Arduino. Esta clase ha sido diseñada para poder desacoplarse en un futuro de Testduino y servir de herramienta de compilación y carga estándar para Arduino.
Dependencias	Library
Atributos	• **includes[]:** listado de cabeceras incluidas en el código fuente del *sketch* (código #include). • **libraries[]:** listado de librerías utilizadas en el *sketch*. • **cpp_path:** ruta absoluta del fichero C++ convertido del *sketch*. • **hex_path:** ruta absoluta del fichero binario hexadecimal generado a partir del *sketch*. • **port:** puerto serie en el cual está conectada la placa Arduino. • **sketch:** *sketch* utilizado. • **path_sdk:** ruta absoluta del Arduino IDE. • **arduino_ide:** versión del Arduino IDE instalado. • **mcu:** modelo de microcontrolador de la placa Arduino.

	• **programmer:** modelo de programador utilizado para la carga. • **fcpu:** frecuencia del microcontrolador de la placa Arduino. • **burnrate:** tasa de carga. • **usb_vid:** identificador del fabricante para USB. • **usb_pid:** identificador del producto para USB. • **variants_path:** ruta absoluta del directorio con la información del hardware según el tipo de placa. • **sketch_temp:** ruta absoluta del directorio temporal de trabajo para el *sketch*. • **tools_path:** ruta absoluta de las herramientas AVRWin Tools. • **ino_path:** ruta absoluta del fichero INO. • **avrdude_configuration_file:** ruta absoluta del fichero de configuración de la herramienta avrdude.
Métodos	• **deleteTemp():** elimina el directorio temporal de trabajo. • **parserSketch():** analiza el *sketch* determinando las librerías utilizadas en él, y realiza la conversión INO a C++. • **compile_():** compila el *sketch*. • **upload():** carga el *sketch* en la placa Arduino. • **compileAndUpload():** realiza la compilación y carga. • **searchLibraries():** busca todas las librerías instaladas en el Arduino IDE. • **searchCorrespondingLibrary(header_include):** busca y devuelve la librería correspondiente a una cabecera incluida en el código fuente. • **addedLibrary(name):** busca y devuelve si una librería ya está siendo utilizada porque ya se encontró con anterioridad. • **debugMsg():** devuelve la parte inicial de un mensaje de *debug*.

Tabla 146: Subcomponente 11

Identificador:	Subcomponente 12
Nombre	Library
Componente	HardwareManagement
Tipo subcomponente	Clase
Función	Representar una librería de Arduino y almacenar información sobre los ficheros que la componen.
Dependencias	Ninguna
Atributos	• **path:** ruta absoluta del directorio de la librería. • **headerfiles[]:** lista de rutas absolutas de los ficheros de cabecera. • **codecppfiles[]:** lista de rutas absolutas de los ficheros C++. • **codeansicfiles[]:** lista de rutas absolutas de los ficheros C.
Métodos	• **directoryRunner():** recorre todo el directorio de la librería buscando ficheros C, C++ y de cabecera y clasificándolos con sus rutas absolutas. • **existsFile(name):** comprueba si existe un fichero en la librería. • **getPath():** devuelve la ruta absoluta del directorio de la librería. • **getHeaderFiles():** devuelve la lista con las rutas absolutas de los ficheros de cabecera de la librería. • **getCppFiles():** devuelve la lista con las rutas absolutas de los ficheros C++ de la librería. • **getCFiles():** devuelve la lista con las rutas absolutas de los ficheros C de la librería.

Tabla 147: Subcomponente 12

4.2.4 Almacenamiento de datos

La solución desarrollada necesita almacenar diversa información. Cada componente software de la solución incorpora su propio sistema de almacenamiento de configuraciones y atributos. En el caso de Jenkins y Subversion, cada uno se gestiona de una determinada manera desarrollada por sus creadores, por lo que no entraremos en detalle.

El programa de adaptación de Testduino requiere de dos ficheros sencillos de texto (formato TXT) para configurarse, estos dos ficheros representan dos tipos de configuración:

- **testconfig:** trata la configuración general de Testduino, que incluye los siguiente aspectos:
    - **Arduino IDE path:** ruta absoluta de Arduino IDE.
    - **Sketches path:** ruta absoluta de los *sketches* de prueba (lugar donde han sido descargados por Jenkins).
    - **Libraries path:** ruta absoluta de las librerías a probar, si las hubiere (lugar donde han sido descargadas por Jenkins).
    - **Baudrate:** tasa de baudios de la comunicación serie.
    - **Success time:** tiempo de espera máximo para que suceda un éxito.
    - **Critical time:** tiempo de espera máximo de monitores pendientes (evita el bloqueo de alguna comunicación que sature el resto del programa). Si se supera el tiempo, se finalizan todos los hilos de monitorización pendientes.
    - **Success keyword:** palabra clave que, al recibirse desde la placa Arduino, simboliza éxito.
    - **Error keyword:** palabra clave que, al recibirse desde la placa Arduino, simboliza error o fallo.
- **testports:** incluye información sobre los puertos a utilizar y el hardware que poseen conectado. Por cada línea (salvo la primera), se incluye un puerto COMX, donde X es el número, y un identificativo del tipo de Arduino conectado:
    - **UNO:** Arduino UNO.
    - **MEGA_ADK:** Arduino Mega ADK for Android.

En ambos ficheros, en cada línea, la configuración va separada por tabulador de su identificador. En *testconfig*, la configuración se introduce entre comillas simples y no es posible eliminar opciones salvo *libraries path*. En *testports*, la primera línea no se debe eliminar nunca.

Otra cuestión de almacenamiento importante son las librerías y *sketches*. Los *sketches* serán descargados por Subversion es una carpeta suya, Testduino, a través de la ruta introducida en su fichero de configuración, los copiará a un directorio interno propio donde trabajar con ellos con los permisos adecuados del sistema. En cada actualización de Jenkins y ejecución, Testduino sobrescribe el directorio interno por completo. Los *sketches* crean un directorio temporal en el momento de su compilación y carga para almacenar ficheros objeto, binarios, etc., que desaparece al terminar dicho proceso.

Las librerías actualizadas deben encontrarse en el directorio *libraries* del Arduino IDE para que la compilación funcione correctamente.

4.3 Entorno de desarrollo

En este apartado, se especifican los entornos de desarrollo integrados utilizados, así como las extensiones auxiliares y librerías que hayan sido necesarias.

4.3.1 Librerías, soporte y lenguajes

Según se expuso en la descripción de la plataforma, el programa de adaptación Testduino se encuentra implementado en lenguaje Python 3, utilizando la versión 32 bits, siendo ésta compatible con 32 bits y 64 bits indistintamente, por motivos relacionados con librerías que se explican posteriormente. La elección de Python fue motivada por las facilidades que otorga en el manejo de cadenas de caracteres, ficheros e implementación rápida, además de ser libre de compilaciones al tratarse de un lenguaje interpretado, y por lo tanto, su instalación es necesaria para la ejecución.

Testduino necesita de dos librerías que Python 3 no incluye de serie. Estas librerías, ya mencionadas en el apartado de plataformas, son PySerial y PyWin32, ambas en su versión 32 bits y que deben instalarse en el sistema para que el intérprete Python pueda utilizarlas. Por diversos problemas de implementación y errores de estas librerías en 64 bits, y al tener que instalarse manualmente de forma anacrónica, se ha tenido que restringir toda la parte de Python a 32 bits hasta que, en futuras versiones, se solucionen dichos problemas.

Por otra parte, la librería Testduo y su *sketch* para la monitorización han sido desarrollados en C++ siguiendo las directrices del lenguaje y del API de programación de Arduino y su microcontrolador.

4.3.2 Software de desarrollo

Para el desarrollo de Testduino se ha utilizado el siguiente software:

Ilustración 37: Logotipo de Eclipse IDE

- **Eclipse IDE:** entorno de desarrollo libre que, en un primer momento, se encuentra orientado para Java, aunque es expandible a otros lenguajes y plataformas mediante extensiones. Soporta compilación y dispone de modo de depuración. El programa de adaptación Testduino ha sido desarrollado completamente en este IDE. Las extensiones utilizadas han sido las siguientes:
    - **PyDev:** convierte Eclipse IDE en un entorno de desarrollo para Python. Añade la compatibilidad necesaria e introduce PyDebug como modo de depuración para Python.

Ilustración 38: Logotipo de Notepad++

- **Notepad++:** software de edición de texto sin formato con opciones avanzadas orientadas a la programación. Ha sido utilizado para la creación de la librería Testduo en C++.

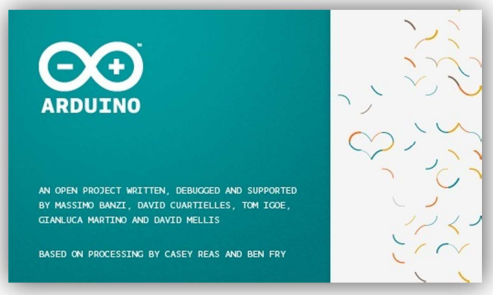


Ilustración 39: Mensaje de bienvenida de Arduino IDE

- **Arduino IDE:** entorno de desarrollo oficial de la plataforma Arduino. Soporta la compilación y carga de software en una placa hardware Arduino, no así la depuración. Se ha utilizado para la escritura del *sketch* de monitorización, en C++, del modo Testduo.

A falta de sistema de depuración en el software que concierne a la programación de Arduino en el modo Testduo, la depuración se ha resuelto utilizando una placa hardware Arduino y con una detección manual de errores alternando diversas posibilidades y variantes.

4.3.3 Documentación del código

Tanto en Testduino como en la librería Testduo y su *sketch*, el código se encuentra debidamente comentado siguiendo los patrones que se mencionan a continuación:

- **Sintaxis Sphinx con RST:** es una solución que combina Sphinx como analizador-generador de documentación automático y RST (reStructuredText) como sintaxis de marcado y analizado perteneciente al paquete Docutils. Permite indicar comentarios de parámetros, retornos y tipos en las funciones. Las clases y líneas de código han sido comentadas utilizando el procedimiento estándar de Python.

- **Sintaxis Javadoc:** la librería ha sido comentada utilizando una adaptación de la sintaxis de Javadoc, pudiendo ser analizada por generadores automáticos como Doxygen.

4.4 Matrices de trazabilidad

En este apartado, se incluyen las matrices de trazabilidad. Gracias a estas matrices, se puede tener constancia del origen de un requisito/funcionalidad y cómo ha ido evolucionando en el diseño hasta ser implementado por un componente.

4.4.1 Trazabilidad SR-UR

A continuación, se muestra la trazabilidad existente entre los requisitos software y los requisitos de usuario.

	CA-UR-01	CA-UR-02	CA-UR-03	CA-UR-04	CA-UR-05	CA-UR-06	CA-UR-07	CA-UR-08	CA-UR-09	CA-UR-10	CA-UR-11	CA-UR-12	CA-UR-13	CA-UR-14	CA-UR-15	CA-UR-16
F-SR-01			X													
F-SR-02	X				X											
F-SR-03	X				X											
F-SR-04		X														
F-SR-05		X														
F-SR-06		X														
F-SR-07		X														
F-SR-08					X											
F-SR-09					X											
F-SR-10					X											
F-SR-11				X												
F-SR-12				X												
F-SR-13				X												
F-SR-14				X												
F-SR-15				X												
F-SR-16				X												
F-SR-17				X												
F-SR-18				X												
F-SR-19				X					X							
F-SR-20				X												
F-SR-21						X										
F-SR-22						X	X									
F-SR-23					X										X	
F-SR-24							X								X	
F-SR-25							X	X		X				X	X	
F-SR-26										X				X		
F-SR-27								X						X		
F-SR-28									X					X		X
F-SR-29									X							
F-SR-30									X		X					
F-SR-31									X							X

Tabla 148: Matriz de trazabilidad F-SR y CA-UR (de F-SR-01 a F-SR-31)

	CA-UR-01	CA-UR-02	CA-UR-03	CA-UR-04	CA-UR-05	CA-UR-06	CA-UR-07	CA-UR-08	CA-UR-09	CA-UR-10	CA-UR-11	CA-UR-12	CA-UR-13	CA-UR-14	CA-UR-15	CA-UR-16
F-SR-32											X				X	
F-SR-33													X			
F-SR-34															X	
F-SR-35											X					
F-SR-36												X			X	
F-SR-37											X	X				
F-SR-38												X				

Tabla 149: Matriz de trazabilidad F-SR y CA-UR (de F-SR-32 a F-SR-38)

	RE-UR-01	RE-UR-02	RE-UR-03	RE-UR-04	RE-UR-05	RE-UR-06	RE-UR-07	RE-UR-08	RE-UR-09	RE-UR-10	RE-UR-11	RE-UR-12	RE-UR-13	RE-UR-14	RE-UR-15	RE-UR-16	RE-UR-17	RE-UR-18
NF-IC-01	X																	
NF-IC-02	X	X																
NF-IC-03		X																
NF-IC-04	X	X																
NF-IC-05		X																
NF-IC-06		X																
NF-IC-07		X				X												
NF-IC-08			X															
NF-IC-09			X															
NF-IC-10				X														
NF-IH-01					X													
NF-IH-02					X													
NF-IH-03						X												
NF-IH-04							X											
NF-IS-01								X										
NF-IS-02								X										
NF-IS-03									X									
NF-IS-04									X									
NF-IS-05								X										
NF-IS-06								X										
NF-IS-07										X								
NF-IS-08											X							
NF-IS-09											X							
NF-IS-10											X							
NF-IS-11												X						
NF-IS-12												X						
NF-IU-01													X					

Tabla 150: Matriz de trazabilidad NF-SR y RE-UR (de NF-IC-01 a NF-IU-01)

	RE-UR-01	RE-UR-02	RE-UR-03	RE-UR-04	RE-UR-05	RE-UR-06	RE-UR-07	RE-UR-08	RE-UR-09	RE-UR-10	RE-UR-11	RE-UR-12	RE-UR-13	RE-UR-14	RE-UR-15	RE-UR-16	RE-UR-17	RE-UR-18
NF-IU-02														X				
NF-IU-03															X			
NF-PO-01																X		
NF-PO-02																X		
NF-SE-01																	X	
NF-SE-02																	X	
NF-TM-01																		X
NF-TM-02																		X
NF-TM-03																		X

Tabla 151: Matriz de trazabilidad NF-SR y RE-UR (de NF-IU-02 a NF-TM-03)

4.4.2 Trazabilidad SR-Componentes

Por último, se muestra la trazabilidad habida entre los requisitos software y los diferentes componentes que forman la solución.

	Componente 1	Componente 2	Componente 3	Componente 4	Componente 5	Componente 6
F-SR-01	X	X				
F-SR-02				X		
F-SR-03			X			X
F-SR-04		X				
F-SR-05		X				
F-SR-06		X				
F-SR-07		X				
F-SR-08		X				
F-SR-09		X				
F-SR-10		X				
F-SR-11		X				
F-SR-12		X	X			
F-SR-13			X		X	X
F-SR-14			X		X	
F-SR-15			X		X	
F-SR-16			X		X	
F-SR-17			X			
F-SR-18		X				
F-SR-19		X	X			

Tabla 152: Matriz de trazabilidad SR y componentes (de F-SR-01 a F-SR-19)

	Componente 1	Componente 2	Componente 3	Componente 4	Componente 5	Componente 6
F-SR-20			X		X	
F-SR-21				X		
F-SR-22				X		
F-SR-23					X	
F-SR-24			X			
F-SR-25				X	X	
F-SR-26				X		
F-SR-27				X		X
F-SR-28			X			
F-SR-29			X			
F-SR-30			X		X	
F-SR-31			X			
F-SR-32						X
F-SR-33			X			
F-SR-34					X	
F-SR-35					X	
F-SR-36					X	
F-SR-37						X
F-SR-38						X
NF-IC-01			X			
NF-IC-02			X			X
NF-IC-03			X		X	
NF-IC-04			X			
NF-IC-05			X		X	
NF-IC-06			X			
NF-IC-07			X			
NF-IC-08		X				
NF-IC-09		X	X			
NF-IC-10	X	X				
NF-IH-01			X			
NF-IH-02			X			
NF-IH-03			X	X		
NF-IH-04			X	X		
NF-IS-01			X			
NF-IS-02			X			
NF-IS-03			X			
NF-IS-04			X			
NF-IS-05		X				

Tabla 153: Matriz de trazabilidad SR y componentes (de F-SR-20 a NF-IS-05)

	Componente 1	Componente 2	Componente 3	Componente 4	Componente 5	Componente 6
NF-IS-06		X				
NF-IS-07						X
NF-IS-08					X	
NF-IS-09					X	
NF-IS-10					X	
NF-IS-11			X			
NF-IS-12			X		X	
NF-IU-01		X				
NF-IU-02		X				
NF-IU-03			X			
NF-PO-01			X			
NF-PO-02			X			
NF-SE-01				X		
NF-SE-02				X		
NF-TM-01			X			X
NF-TM-02			X			X
NF-TM-03			X			X

Tabla 154: Matriz de trazabilidad SR y componentes (de NF-IS-06 a NF-TM-03)

5. Pruebas y evaluación

En este apartado, se incluyen la especificación de las pruebas que aseguran el correcto funcionamiento del programa de adaptación de Testduino, así como la evaluación y el estudio del rendimiento bajo diferentes configuraciones, observando cómo afecta al procesamiento general del microcontrolador de Arduino.

5.1 Pruebas de verificación

Las pruebas que se ejecutan tienen como objetivo la comprobación de la estabilidad y el correcto funcionamiento requerido del programa de adaptación de Testduino, el componente más crítico de la solución. De esta manera, se valida y verifica la solución, ya que el resto de componentes poseen sus propias validaciones realizadas por sus desarrolladores respectivos, al tratarse de software de terceros en versión estable.

5.1.1 Requisitos previos

Es necesario tener en cuenta unos requisitos previos para poder efectuar el plan de pruebas. Por lo tanto, se requieren unas tareas previas de preparación que asegurarán su correcta ejecución:

- **Definición de las pruebas:** las pruebas deben ser definidas, indicando sus factores de entorno.

- **Especificación del criterio de aceptación:** detallará cuándo una prueba es considerada como aceptada y cuándo no.

- **Determinación de las salidas de las pruebas y su posterior calificación:** se detallan las salidas esperadas así como la calificación obtenida finalmente a la hora de finalizar la prueba.

5.1.2 Entorno de prueba

En este apartado, se detalla el entorno de pruebas utilizado, que debe ser equivalente, en la medida de lo posible, al entorno en producción, ya que el resultado esperado debe ser exactamente el mismo. El entorno de prueba, tanto de Jenkins como del programa de adaptación de Testduino (sobre el cual se centran estas pruebas), está compuesto por la siguiente configuración:

- Un mínimo de 2 GB de memoria RAM.
- Un procesador Intel Pentium D 965 o superior con 15 GFLOPs de potencia mínima.
- Un espacio libre en disco duro de al menos 10 GB.
- Sistema operativo Windows 7 x32 o x64.
- Python 3.x, PySerial y PyWin32 instalados.
- Última versión de Jenkins instalada.
- Arduino IDE 1.0.2 instalado.
- Conexión a internet de 1 Mb como mínimo con acceso a repositorio Subversion.
- 1 placa Arduino Mega ADK como mínimo.
- 2 placas Arduino UNO como mínimo.

5.1.3 Criterio de aceptación

En este apartado, se establece un criterio de aceptación de las pruebas llevadas a cabo. Este criterio es recogido en estos puntos:

- Cada una de las pruebas deben ejecutarse siguiendo **el orden prestablecido** en el documento presente.

- Debido a la gran variedad de combinaciones existentes, las pruebas detallarán **aspectos genéricos** que deben cumplirse en cualquier caso, sin entrar a razonar aspectos específicos que pueden ser influenciables por factores externos.

- Las pruebas únicamente se llevarán a cabo de la forma que se describen, cumpliendo con el **procedimiento establecido**, de forma que, aunque existiendo otras vías de ejecución, no se probarán siempre y mientras no se encuadren dentro del procedimiento descrito en la prueba.

5.1.4 Especificación de los casos de prueba

En este apartado, se especifican los casos de prueba, cuyo objetivo es asegurar el correcto cumplimiento de las pruebas.

Estos casos son detallados según la siguiente plantilla:

Identificador	CP-XX
Descripción	
Especificaciones de entrada	
Especificaciones de salida	
Necesidades de entorno	

Tabla 155: Plantilla de Prueba

- **Identificador:** identificador del caso de prueba. Consiste en un código alfanumérico formado por la cadena CP (*caso de prueba*), seguido de dos dígitos variables “XX”.

- **Descripción:** describe de manera breve en que consiste el caso de prueba.

- **Especificaciones de entrada:** refiere las entradas necesarias para la realización de la prueba. Contiene los ficheros, parámetros de entrada y configuración necesaria.

- **Especificaciones de salida:** refiere los resultados esperados de la prueba.

- **Necesidades de entorno:** requisitos previos a la ejecución de la prueba que se tienen que dar de forma que se pueda ejecutar la prueba en unas características óptimas. Dichas necesidades de entorno serán configuraciones del sistema, ficheros y estado del sistema.

A continuación, se detallan las pruebas realizadas:

Identificador	CP-01
Descripción	Se prueba que el programa puede testear *sketches* libremente en las placas tipo UNO o MEGA_ADK indistintamente. Se asegura la correcta compilación, carga y monitoreo, induciendo fallos que deben ser detectados por el programa.
Especificaciones de entrada	Ejecución con varios *sketches* de diferente complejidad y tres Arduino: 2 UNO y 1 MEGA_ADK.
Especificaciones de salida	Se espera que todos los resultados den positivo en el testeo de los *sketches*.
Necesidades de entorno	• 2 Arduino UNO conectados. • 1 Arduino MEGA_ADK conectado. • 3 *sketches* sin librerías asociadas. • 3 *sketches* con librerías asociadas.

Tabla 156: CP-01

Identificador	CP-02
Descripción	Se prueba que el programa puede ejecutar el modo Testduo (sólo para Arduino UNO). Se comprueba el correcto funcionamiento del modo Testduo (protocolos, *sketch* de monitorización, etc.) induciendo fallos a propósito.
Especificaciones de entrada	Ejecución con un *sketch* en modo Testduo.
Especificaciones de salida	Se espera que todos los resultados den positivo en el testeo de los pines del Arduino UNO.
Necesidades de entorno	• 2 Arduino UNO conectados. • 1 *sketch* en modo Testduo.

Tabla 157: CP-02

Identificador	CP-03
Descripción	Se prueba que el programa puede ejecutar *sketches* en puertos concretos, junto con *sketches* libres y *sketches* en modo Testduo. De esta manera, se asegura que el flujo de ejecución intercalando diferentes configuraciones de *sketches* es correcto, verificando la estabilidad del programa.
Especificaciones de entrada	Ejecución de conjunto de *sketches* de diferentes complejidades en puertos asociados, libres y en modo Testduo.
Especificaciones de salida	Se espera que todos los resultados den positivo en el testeo de los pines del Arduino UNO, y que el resultado de ejecución de los *sketches* sea correcto.
Necesidades de entorno	• 2 Arduino UNO conectados. • 1 Arduino MEGA_ADK conectado. • 2 *sketches* en modo Testduo. • 2 *sketches* en puerto específico. • 2 *sketches* libres sin puerto específico.

Tabla 158: CP-03

Identificador	CP-04
Descripción	Se comprueba el soporte de *sketches* de diferentes librerías asociadas y múltiples objetos (con mucha complejidad y retardos).
Especificaciones de entrada	Ejecución de conjunto de *sketches* de alta complejidad y ocupación en memoria.
Especificaciones de salida	Se espera que todos los resultados den positivo en el testeo de los *sketches*.
Necesidades de entorno	• 1 Arduino UNO conectado. • 1 Arduino MEGA_ADK conectado. • 4 *sketches* con más de 2 objetos y 2 librerías asociadas.

Tabla 159: CP-04

Identificador	CP-05
Descripción	Se prueba el soporte de diferentes anchos de banda en la transmisión serie entre Arduino UNO o Mega ADK y el programa.
Especificaciones de entrada	Ejecución de un conjunto de *sketches* varias veces cambiando la tasa de baudios en cada ejecución.
Especificaciones de salida	Se espera que todos los resultados den positivo en el testeo de los *sketches* sin problemas de lectura de monitorización.
Necesidades de entorno	• 1 Arduino UNO conectado. • 1 Arduino MEGA_ADK conectado. • 4 *sketches* libres.

Tabla 160: CP-05

Identificador	CP-06
Descripción	Se prueba el soporte de diferentes anchos de banda en la transmisión utilizando el modo Testduo (lectura y escritura en puerto serie).
Especificaciones de entrada	Ejecución de un *sketch* en modo Testduo con diferentes tasas de baudios en cada ejecución (necesario actualizar la tasa de baudios configurada en la librería Testduo).
Especificaciones de salida	Se espera que todos los resultados den positivo en el testeo de los *sketches* sin problemas de comunicación entre las placas Arduino y el programa.
Necesidades de entorno	• 2 Arduino UNO conectados. • 1 *sketch* en modo Testduo.

Tabla 161: CP-06

Identificador	CP-07
Descripción	Se prueba el correcto tratamiento de errores de comunicación (de puerto serie) realizado por el programa.
Especificaciones de entrada	Ejecución de varios *sketches* con varias placas Arduino cuyo puerto se encuentre inaccesible por cualquier razón (incluida la configuración incorrecta).
Especificaciones de salida	Se espera que el programa ignore las pruebas en puertos con problemas.
Necesidades de entorno	• 2 Arduino UNO conectados. • 1 Arduino MEGA_ADK conectado. • 5 *sketches* diferentes modos (puerto específico y Testduo).

Tabla 162: CP-07

Identificador	CP-08
Descripción	Se prueba la actualización de la librería a probar en el directorio de librerías del IDE de Arduino.
Especificaciones de entrada	Ejecución de varios *sketches* que utilicen una determinada librería que ha sido descargada del repositorio de versiones.
Especificaciones de salida	Se espera que el programa sustituya la anterior versión (si existe) y ejecute las pruebas correctamente.
Necesidades de entorno	• 2 Arduino UNO conectados. • 5 *sketches* que utilicen una librería asociada en pruebas.

Tabla 163: CP-08

5.2 Evaluación y análisis de resultados

Las evaluaciones propuestas evalúan aspectos considerados influyentes por el programa de adaptación de Testduino, como, por ejemplo, el rendimiento del hardware y su integración con el software, así como aspectos de proceso de la propia placa Arduino, sobre todo el impacto del modo Testduo sobre ella. Estos aspectos son los siguientes:

- **Tiempos de preparación** (compilación y carga) del *sketch* sobre los diferentes tipos de placa Arduino soportadas, en **comparación con el IDE oficial** de Arduino. De esta manera, podremos observar demoras o mejoras de rendimiento en Testduino respecto al IDE oficial.

- **Ejecución de *sketches* de estudio de rendimiento (*benchmarks*)** para la evaluación de los diferentes tipos de placas Arduino con diferentes microcontroladores, en diferentes aspectos: asignaciones de memoria, operaciones matemáticas, etc.

- **Tiempos de prueba** de Testduino en base a diferentes **velocidades de transmisión** del puerto serie (medido en baudios). De esta forma, podremos deducir el grado de influencia del ancho de banda en la dinámica de ejecución de las pruebas.

- **Cantidad de memoria** del microcontrolador consumida por *sketches* de diversos tipos en base a la utilización de la librería de **Testduo** y la instanciación de sus objetos.

5.2.1 Entorno de evaluación

El rendimiento del programa de adaptación de Testduino, excluyendo los factores dependientes del hardware, y el resto de componentes como Jenkins dependen, en buena medida, de la capacidad de procesamiento y velocidad del equipo utilizado para su evaluación. En esta evaluación, el objetivo es establecer conclusiones en la relación entre el programa de adaptación de Testduino y las placas Arduino, y por lo tanto, se detalla a continuación las características del entorno utilizado:

- Intel Core i7-2670QM de 4 núcleos a 2,20 GHz con *TurboBoost* a 3,10 GHz y tecnología *HyperThreading* (8 núcleos virtuales).
- 16 GB de memoria RAM.
- Disco duro SSD Intel SA2CW120G3 de 128 GB para el sistema operativo.
- Disco duro de 750 GB Western Digital WD7500BPKT a 7200 RPM para el software de la solución y los *sketches*.
- GPU nVIDIA GeForce GTX 560M con 2 GB dedicados de video.
- Sistema operativo Microsoft Windows 8 Pro de 64 bits.

- Comunicación entre las placas y el sistema mediante estándar USB 2.0.

Las placas Arduino utilizadas son las soportadas por la solución: Arduino UNO y Arduino Mega ADK for Android. Sus características técnicas se encuentran descritas en el apartado 4.1 Hardware y software empotrado.

Los *sketches* utilizados son:

- *Blink*: *sketch* más básico y sencillo. No utiliza librerías externas, y su complejidad consiste en encender y apagar un LED de forma intermitente.

- Testduo: *sketch* específico construido para la prueba del modo Testduo, realiza hasta cinco comunicaciones de trazas de Testduo con diferentes valores digitales y de PWM. Incorpora una librería ligera (Testduo) con cinco objetos instanciados. Evidentemente, la utilización de este sketch implica la carga y compilación y ejecución del *sketch* de monitorización en otra placa Arduino UNO. En la evaluación, según el tipo, se realizarán variaciones del *sketch* para medir diferentes aspectos.

- Testduo sin capacidades Testduo: se utiliza el *sketch* anterior sólo para observar su tiempo de compilación y carga, sin capacidad de ejecución Testduo. Es equivalente a utilizar un *sketch* con una librería ligera y cinco objetos instanciados.

- *GSM Web Client*: *sketch* que representa un *sketch* de uso real con una librería pesada (librería GSM3 de Telefónica I+D) de cinco cabeceras incluidas y cinco objetos instanciados.

- *Benchmark* de *Mikrocontroller*: *sketch* que mide rendimiento del microcontrolador en asignaciones de memoria de diferentes tipos de datos (INT16, INT32 y FLOAT) y operaciones matemáticas con ellos (suma, resta, multiplicación, división, seno, etc.). Es una adaptación de un programa C estándar realizado para medir el rendimiento de diversos microcontroladores en general[56].

5.2.2 Análisis de resultados

En este apartado, se procede a detallar en profundidad los resultados obtenidos y se razona la posible causa de los mismos.

5.2.2.1 Compilación y carga

En esta evaluación, se mide el tiempo de preparación del *sketch* para su ejecución. El tiempo de preparación es influido por diferentes factores:

- Tiempo de compilación:

    - Complejidad y tamaño del *sketch*.
    - Número de librerías utilizadas en él.
    - Grado de utilización.
    - Complejidad y tamaño de las librerías.
    - Capacidad de procesamiento del equipo.
    - Velocidad de procesamiento del equipo.
    - Memoria disponible en el equipo.

- Tiempo de carga:
    - Tamaño del hexadecimal resultante.
    - Velocidad de la comunicación serie para la escritura en memoria (burnrate).

Estos factores, como se puede observar, son variables y dependen del equipo utilizado y del microcontrolador de la placa. En concreto, la carga es muy decisiva, ya que diferentes microcontroladores soportan una variedad de velocidades de carga determinadas. Por la parte de la compilación, dicho proceso se encuentra dividido en varias fases, como se indica en el apartado 4.1.3 Compilación, y su resultado será el total de tiempo que requiere cada fase, es decir, cada programa utilizado en la compilación.

Observamos los datos obtenidos, tiempo en segundos, realizando el proceso con el IDE de Arduino versión 1.0.2:

	UNO	Mega ADK
Blink	4,17 s	3,87 s
Testduo*	5,85 s	5,62 s
GSM Web Client	13,31 s	12,54 s

Tabla 164: Tiempo de preparación con Arduino IDE

Y a continuación, se detalla el resultado realizando el mismo proceso a través del programa de adaptación de Testduino:

	UNO	Mega ADK
Blink	4,10 s	4,32 s
Testduo*	6,29 s	6,38 s
GSM Web Client	13,86 s	12,94 s

Tabla 165: Tiempo de preparación con Testduino

** Modificación sin capacidades Testduo, sólo para compilación y carga, ya que el Arduino Mega ADK no es soportado en modo Testduo.*

El ancho de banda utilizado en el proceso de carga es, en todos los casos mostrados, de 115.200 baudios, lo máximo que permite la comunicación serie con Arduino, y el estándar utilizado por el IDE oficial. Se puede apreciar una diferencia en los tiempos entre los dos tipos de placas Arduino soportadas, siendo favorable en rapidez al Arduino Mega ADK en el IDE oficial, aunque esta afirmación es contradicha en la prueba con el programa de adaptación de Testduino.

Por otra parte, parece ser que el programa de adaptación de Testduino perjudica el rendimiento en favor del IDE oficial, claro está que, lógicamente, ambos software no están basados en la misma plataforma, ya que, el IDE de Arduino está programado en Java, y el programa de adaptación está implementado con Python. Python es un lenguaje interpretado muy rápido para *scripting*, pero que puede penalizar la complejidad de un paradigma orientado a objetos y la utilización de librerías externas para acceder a servicios del sistema operativo, mientras que Java cuenta con una máquina virtual optimizada por Oracle que proporciona acceso a dichos servicios sin depender de librerías de terceros. La gestión del proceso y el proceso en sí es idéntica en ambos casos.

Para profundizar en estos aspectos, se presentan las siguientes comparativas:

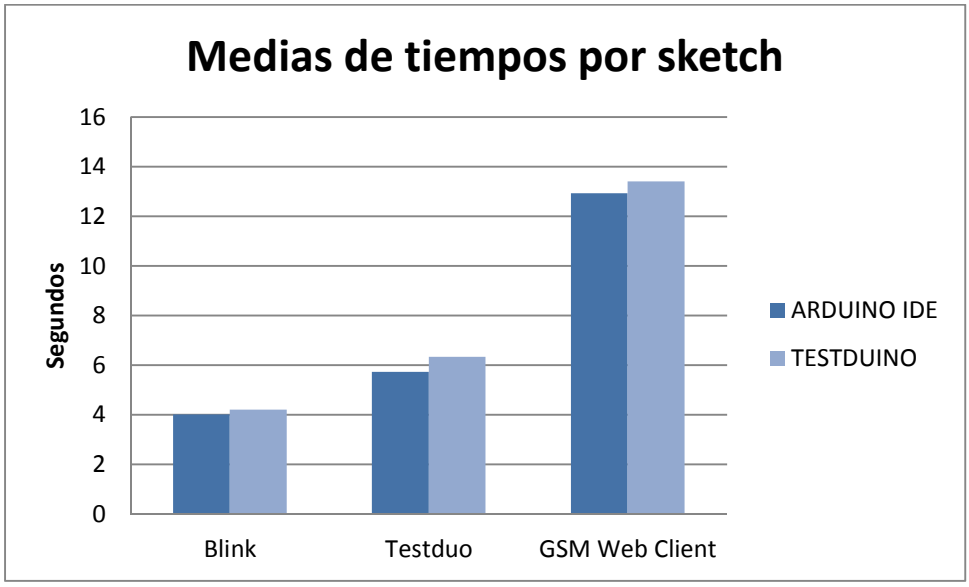


Ilustración 40: Gráfica de medias de tiempos de compilación y carga por sketch

Se puede apreciar, a parte del mayor tiempo consumido por Testduino (en torno a los 10 segundos más de media manteniéndose constante en todos los casos), el aumento del tiempo producido por la complejidad del *sketch* y la cantidad de objetos instanciados de librerías incluidas.

Al incorporar una librería, la instanciación de un objeto de alguna de sus clases requiere reservar espacio en memoria, dependiendo de la cantidad de métodos y atributos incorporados en la clase. Esto afectará al rendimiento de la memoria, como se muestra más adelante, pero también afecta al proceso de compilación y de enlace, teniendo que enlazar objetos al código principal del *sketch*, y reservando la correspondiente lógica descrita en el hexadecimal resultante. El aumento de tamaño de bytes del hexadecimal provoca un mayor tiempo de carga.

Podemos comprobar que la diferencia entre un *sketch* sin librerías asociadas y un *sketch* con una librería pero cinco objetos iguales instanciados apenas representa un aumento significativo del tiempo, ya que no deja de ser la misma lógica, que es enlazada por cada objeto instanciado a través de los punteros de las funciones. Por el contrario, cuando se requieren objetos de diferentes clases, el aumento de tiempo será resultado del tiempo de compilación de una clase C++ más el tiempo empleado por el enlazador en enlazar dichos objetos.

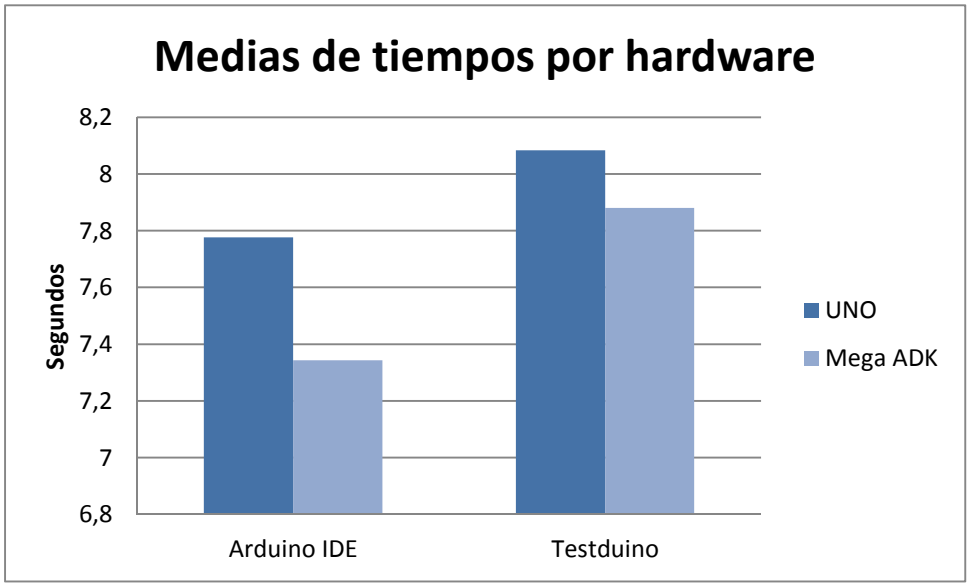


Ilustración 41: Gráfica de medias de tiempos de compilación y carga por hardware

En la gráfica anterior, se aprecia una fuerte mejora de los tiempos de carga y compilación para el Arduino Mega ADK, frente a los del Arduino UNO. Por otra parte, resulta interesante el retardo que introduce el programa de adaptación de Testduino, llegando incluso a intentar igualar los tiempos como resultado del superior aumento de dicho retardo en el Arduino Mega ADK respecto al Arduino UNO.

Podemos deducir que el proceso de compilación es más eficiente para el microcontrolador ATMEGA2560 con GCC y las herramientas GNU que para el ATMEGA328. El proceso de carga sigue la misma velocidad de transmisión en ambos casos, y podría verse afectado por una mayor velocidad de respuesta del microcontrolador ATMEGA2560 debido a la mayor cantidad de memoria, aunque el aumento debería ser ínfimo y tener poco efecto en el resultado total. Aun así, los factores influyentes deben encontrarse entre el proceso de compilación y la comunicación, repasemos estas fases más detenidamente:

- **Comunicación:** ambos utilizan un microcontrolador aparte para la conversión FTDI a serie. En el Arduino Mega ADK se encuentra un ATMEGA8U2, y en el Arduino UNO un modelo ATMEGA16U2. Ambos son microcontroladores de 8 bits y, aunque el ATMEGA16U2 es una evolución del ATMEGA8U2, la evolución solo se presenta en cuestiones de memoria, porque la velocidad de reloj sigue siendo 16 MHz, y con lo cual, vistos los resultados, no puede afectar positivamente al Arduino Mega ADK.

Microcontrolador	ATMEGA16U2	ATMEGA8U2
Placa Arduino	Arduino UNO	Arduino Mega ADK
Voltaje de operación	5 V	5 V
E/S	22	22
Entradas analógicas	16	16
Memoria *flash*	16 KB	8 KB
SRAM	4 KB	512 bytes
EEPROM	4 KB	512 bytes
Velocidad de reloj	16 MHz	16 MHz

Tabla 166: Comparativa entre microcontroladores de comunicación

- **Compilación:** esta fase parece ser el detonante del mayor tiempo del Arduino UNO. Se han realizado pruebas de sólo compilación utilizando el IDE oficial y el *sketch* más complejo, y los resultados han sido:

	Arduino UNO	Arduino Mega ADK
GSM Web Client	11,70 s	7,30 s

Tabla 167: Comparativa de tiempos de compilación entre UNO y Mega ADK

Por otra parte, el tiempo superior del programa de adaptación de Testduino, tiene que estar relacionado con la plataforma sobre la que se ejecuta el código fuente, como se ha introducido anteriormente. Arduino IDE es un programa Java que se ejecuta sobre una máquina virtual que tiene accesos casi directos a servicios del sistema operativo, al estar más preparado para aplicaciones de escritorio con orientación a objetos. En cambio, el programa de adaptación de Testduino es un programa Python orientado a objetos. Si bien Python permite este paradigma, su punto fuerte es el *scripting*, aunque la verdadera causa está relacionada con la gestión de los recursos del sistema operativo, a parte de la rapidez del propio intérprete de Python en Microsoft Windows, ya que es necesario utilizar dos librerías externas desarrolladas por terceros para la comunicación serie y el acceso a la API de Microsoft Windows, que no poseen la mejor garantía de rendimiento.

5.2.2.2 Rendimiento de los microcontroladores

En esta evaluación, se someten ambos microcontroladores soportados por el programa de adaptación de Testduino a un *benchmark*.

Debido a que Arduino utiliza su propia derivación de C con *Processing*, ha sido necesario realizar una adaptación de un *benchmark* existente para microcontroladores, aportado por un miembro de la comunidad del sitio web *Mikrocontroller*. Los cambios introducidos en la forma de proceder son mínimos y no afectan al test, tratándose sólo de cambios de estructura de programa para poder ser compilado.

Este *benchmark* realiza mediciones en las siguientes operaciones:

- **Eliminación de 256 bytes** de memoria.
- **Relleno de 256 bytes** de memoria.
- **Copia de 256 bytes** entre diferentes posiciones de memoria.
- **Eliminación de 64 palabras de 32 bits.**
- **Relleno de 64 palabras de 32 bits.**
- **Copia de 64 palabras de 32 bits.**
- **División** de enteros de **16 bits.**
- **División** de enteros de **32 bits.**
- **División de coma flotante.**
- **Multiplicación de coma flotante.**

- **Seno** con número de **coma flotante.**
- **Conjunto variado** de operaciones: asignaciones, desplazamientos, sumas, restas, etc.

Los datos obtenidos, medidos en microsegundos y utilizando el IDE oficial de Arduino, son los siguientes:

	Arduino UNO	Arduino Mega ADK
Borrado 256 bytes	744 µs	768 µs
Relleno 256 bytes	472 µs	472 µs
Copia 256 bytes	628 µs	632 µs
Borrado 128 palabras 16 bits	148 µs	152 µs
Relleno 128 palabras 16 bits	148 µs	152 µs
Copia 128 palabras 16 bits	252 µs	252 µs
Borrado 64 palabras 32 bits	92 µs	96 µs
Relleno 64 palabras 32 bits	92 µs	96 µs
Copia 64 palabras 32 bits	164 µs	168 µs
División de INT16	20 µs	20 µs
División de INT32	44 µs	48 µs
División FLOAT	36 µs	36 µs
Multiplicación FLOAT	12 µs	16 µs
SEN de FLOAT	108 µs	112 µs
Conjunto final	111.196 µs	111.232 µs

Tabla 168: Comparativa de rendimiento entre Arduino UNO y Arduino Mega ADK

Para obtener una conclusión más precisa, a la vista de los datos, a continuación se dividen los resultados arrojados mediante las siguientes gráficas.

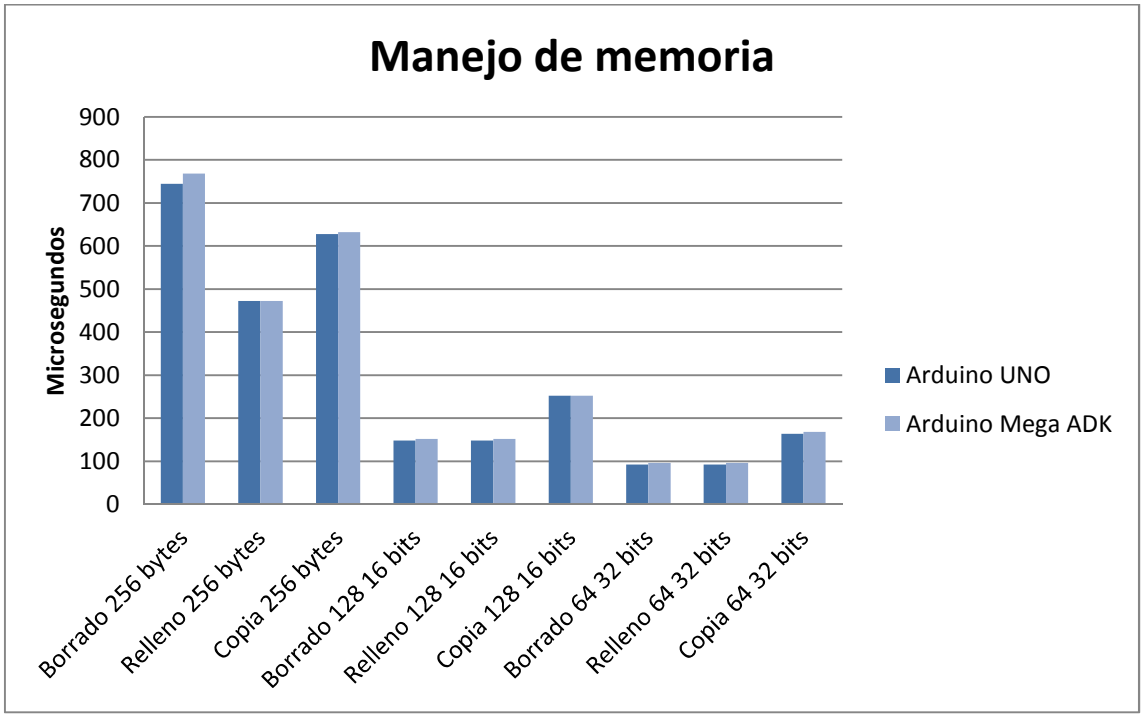


Ilustración 42: Gráfica de tiempos de gestión de memoria

A la vista de los datos, la gráfica no deja lugar a duda de que el Arduino UNO, con su microcontrolador ATMEGA328P es más rápido realizando asignaciones de memoria. Resulta llamativo que su mejor rendimiento se alcance utilizando palabras de 32 bits, siendo el microcontrolador de 16 bits.

El mayor tiempo del Arduino Mega ADK, con el microcontrolador ATMEGA2560 puede deberse a que disponer mayor cantidad de memoria, tanto en *flash* como en *EEPROM* y *SRAM*, penalice la gestión de memoria básica de la arquitectura AVR, obteniendo un minúsculo retardo en el direccionamiento del puntero, ya que el espacio de direcciones de la arquitectura es unificado. El juego de instrucciones es el mismo para ambos microcontroladores y están diseñados para ejecutar los accesos de memoria en muy pocos ciclos de reloj, por lo que, esta hipótesis planteada es la más probable.

En cuanto a los tiempos reducidos de operaciones de memoria con palabras de 16 y 32 bits, el microcontrolador incorpora algunos pares de registros que son usados como punteros de 16 bits al espacio de memoria externa (conocidos como los registros X, Y y Z). Este tipo de diseño agiliza el uso de direcciones de datos de 16 bits y, en su combinación, de 32 bits sin complicar el diseño y optimizando el número de ciclos de reloj.

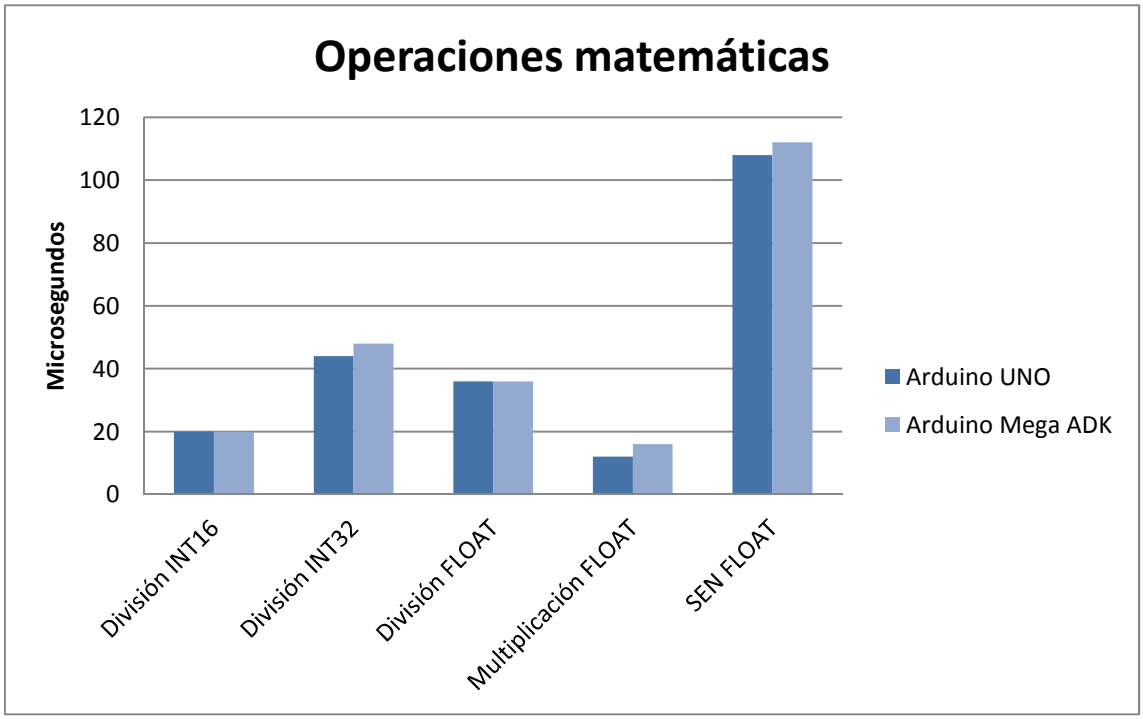


Ilustración 43: Gráfica de tiempos de operaciones matemáticas

En las operaciones matemáticas, también se obtiene una mejora en velocidad por parte del Arduino UNO (ATMEGA328P) respecto al Arduino Mega ADK (ATMEGA2560). Lógicamente, operaciones como las divisiones, y sobretodo el seno, son más complejas para este tipo de arquitectura.

La mayoría de las operaciones básicas aritmético-lógicas establecidas en el juego de instrucciones AVR se realizan en un solo ciclo de reloj, soportando operaciones entre registros o entre una constante y un registro. Esta arquitectura sólo incorpora una unidad aritmético-lógica (ALU) directamente conectada a los registros y al bus principal de acceso a memorias según la arquitectura Von Harvard.

Es muy interesante el rendimiento del juego de instrucciones, en conjunto con el diseño de la arquitectura, en la realización de operaciones de coma flotante, donde se alcanza un mayor rendimiento que en operaciones con enteros de 32 bits. Debido a que el tipo FLOAT ocupa 32 bits de coma flotante con precisión simple, se puede inducir que existe una optimización en la ALU para operaciones de coma flotante, no siendo así para tipos de datos más grandes.

La diferencia de tiempos a favor del Arduino UNO es marginal, y debe ser causa de la gestión de memorias de mayor tamaño en el caso del Arduino Mega ADK, tal y como se ha comentado anteriormente, ya que el diseño y las especificaciones del núcleo AVR son las mismas en ambos microcontroladores.

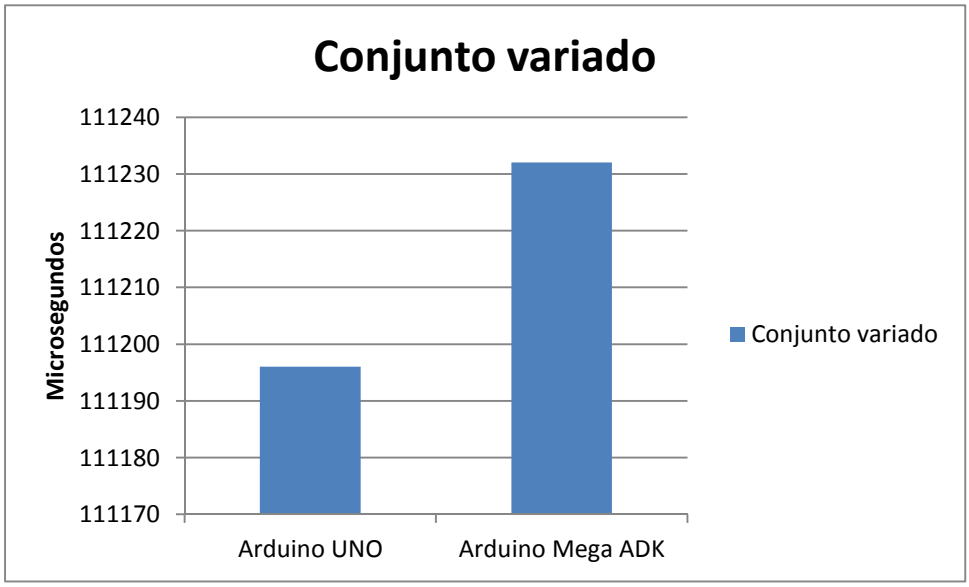


Ilustración 44: Gráfica de tiempo consumido por la ejecución del conjunto variado

En resumen, utilizando diversas operaciones matemáticas y accesos a memoria, en lo que sería un código que exige capacidad al microcontrolador, se observa que la penalización del Arduino MEGA ADK, o en otras palabras, del microcontrolador ATMEGA2560, en los accesos a memoria y en la realización de cálculos conlleva una pérdida de rendimiento que puede llegar a ser importante. Esta pérdida, a la vista de las pruebas realizadas, ronda más del 50% favorable al Arduino UNO con su microcontrolador ATMEGA328P, con menor capacidad de memoria y misma potencia de procesador. Por lo tanto, se concluye que la arquitectura AVR fue diseñada, desde un principio, para un manejo de memoria limitado.

5.2.2.3 Rendimiento de la comunicación serie

En este apartado, se detalla la evaluación de la influencia de la comunicación serie, refiriéndonos al ancho de banda utilizado, en el tiempo de ejecución de los test del programa de adaptación de Testduino.

Este aspecto es crucial en el modo Testduo, en el cual, la comunicación de ambas placas con el equipo es la base de la monitorización, y es el modo donde más se abusa de esta comunicación y con total relevancia en el resultado de los test. Por este motivo, la evaluación se ha realizado ejecutando el *sketch* de ejemplo de Testduo con cinco puntos de monitorización de pines, donde el resultado es correcto en todos.

Según lo especificado, en la prueba se activará el protocolo de comunicación cinco veces, conllevando un total de 20 transmisiones serie entre el equipo y las placas Arduino UNO de entre 10 y 30 bytes o caracteres. Ambas placas Arduino utilizan la misma tasa de baudios de transferencia serie, en conjunto con la configuración del programa de adaptación de Testduino.

La medición realizada consiste en el tiempo de ejecución del test completo de Testduo con el *sketch* mencionado, omitiendo el tiempo de compilación y de carga. Las tasas de baudios empleadas son las indicadas por el monitor serie del IDE oficial de Arduino. Los resultados obtenidos, medidos en segundos, son los siguientes:

Tasa de baudios	Tiempo
9.600	44,32 s
19.200	41,21 s
28.800	40,82 s
57.600	42,86 s
115.200	41,02 s

Tabla 169: Comparativa de rendimiento de la comunicación serie

Es importante tener en cuenta el siguiente aspecto: el tiempo mostrado incluye el tiempo de procesado de la comparativa de pines de Testduino, ya que se mide el rendimiento de una prueba Testduo completa con varios anchos de banda. Los datos son muy similares, y no parece existir una relación de mayor velocidad serie menor tiempo de ejecución. Para realizar un análisis más a fondo, observemos la tendencia de la relación:

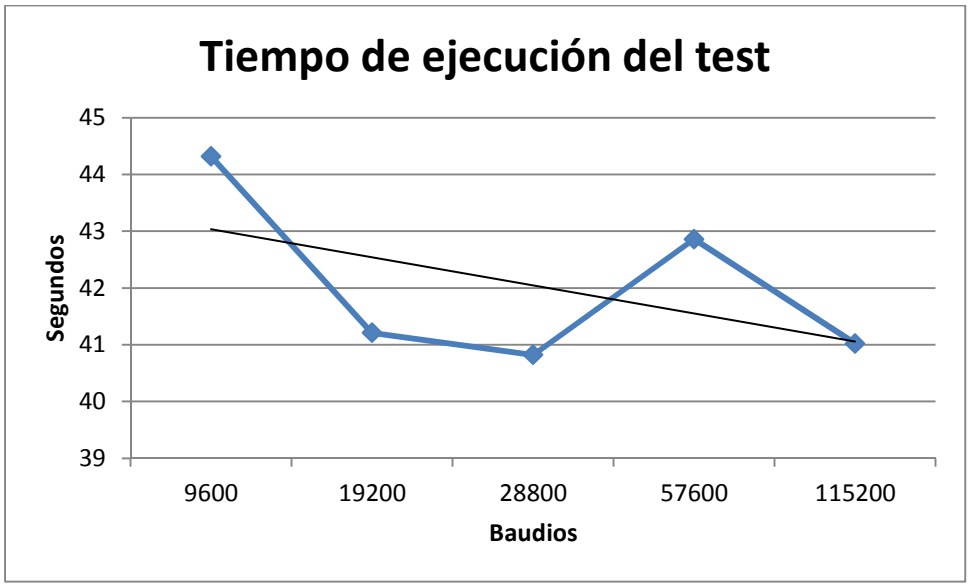


Ilustración 45: Gráfica de tendencia del tiempo de ejecución respecto a los baudios

Aunque se observa una lógica tendencia a la baja en el tiempo, los datos son muy contradictorios respecto a lo que debería ser. Nos encontramos con un momento, en el cual, la tasa de baudios penaliza al tiempo, en vez de mejorarlo, aunque desde una perspectiva general, no existe ninguna mejora.

Este fenómeno puede estar producido por el propio programa de adaptación de Testduino, que al igual que ocurría con la compilación y carga de *sketches*, puede ocasionar retardos que igualen los tiempos por la gestión de la comunicación serie de Python y la librería PySerial. Por lo tanto, se concluye que el programa de adaptación de Testduino no logra optimizarse con una mayor velocidad de transmisión.

5.2.2.4 Impacto de la librería Testduo en los sketches

Los microcontroladores utilizados por la plataforma Arduino son muy limitados en memoria, requiriendo un óptimo uso de la memoria en tareas complejas. Por este motivo, esta evaluación mide el impacto que ejerce la librería Testduo y la instanciación de su clase en los *sketches* que se utilizarán para los test.

El diseño de la librería Testduo es tiene muy baja sobrecarga, consta de una única clase y tres métodos, con un número muy reducido de atributos, organizando los datos de los pines en matrices de hasta 50 pines posibles, ocupando un byte por pin. La medición se basa en calcular el tamaño del fichero hexadecimal final, utilizando el IDE oficial de Arduino debido a que lo indica al finalizar el proceso de compilación, probando con diferente número de objetos Testduo y sin ellos. Debido al uso de matrices de memoria estática predefinidas, hasta el máximo posible de pines definidos, el espacio ocupado será el mismo sin importar el número de pines a monitorizar. El *sketch* utilizado es un *sketch* básico que sólo usa el modo Testduo sin realizar más operaciones ni poseen parámetros ni variables sobrantes que ocupen memoria.

Los datos observados, medidos en bytes ocupados en la memoria *flash* del microcontrolador, son los siguientes:

Nº de objetos	Bytes utilizados
0	2.576
1	5.528
2	5.668
3	5.812
4	5.970

Tabla 170: Comparativa de memoria ocupada por la librería Testduo

Como se puede apreciar, el hexadecimal base sin utilizar la librería Testduo ocupa 2.576 bytes. Existe un crecimiento al utilizar el primer objeto Testduo, ya que se debe incorporar toda la lógica de la librería al hexadecimal y reservar el espacio de memoria para los atributos. En cambio, según se va aumentado el número de instancias de la clase Testduo, se observa que el crecimiento baja drásticamente, ya que es la misma lógica, y el aumento ocurrido estaría justificado por los atributos propios del nuevo objeto, en torno a los 1.500 bytes de crecimiento.

En la siguiente gráfica, se puede ver mejor la tendencia:

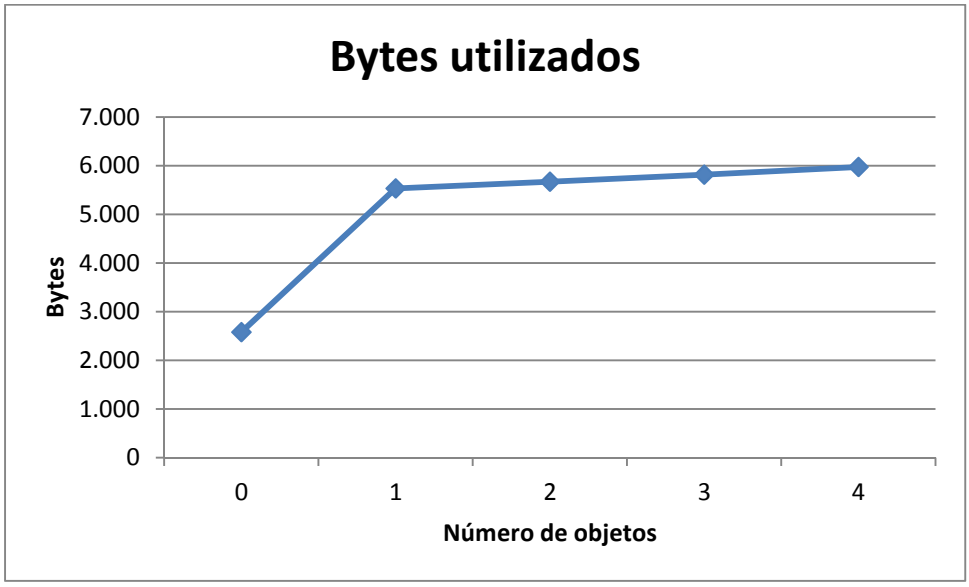


Ilustración 46: Gráfica de tendencia de bytes consumidos por Testduo

En conclusión, la utilización de varios objetos Testduo no supone una pérdida de memoria importante, dado que los *sketch* de Arduino no suelen ser muy complejos, con lo que es poco probable que se necesiten una cantidad de objetos Testduo grande para monitorizar los pines de un *sketch*, contando con que los objetos modificados se pueden reutilizar.

Por otra parte, si restamos los bytes base ocupados por el *sketch* en sí, nos quedamos con un aumento de 2.952 bytes.

6. Planificación

En este apartado, se incluye una planificación detallada del proyecto, así como una valoración sobre la estimación de costes.

6.1 Estimación con COCOMO II

A la hora de desarrollar un proyecto, suele ser necesario estimar el coste del mismo para la toma de decisiones en diversas áreas de interés, tales como recursos tecnológicos o recursos humanos necesarios.

En nuestro caso, la estimación no se ha producido, ya que la naturaleza del proyecto y el software desarrollado en él no es el más adecuado ni se adapta a este tipo de modelos de estimación.

La utilización de lenguajes de *scripting* como Python y la fuerte relación con el hardware de microcontroladores no contribuyen a realizar una estimación realista, ya sea por puntos de función o por líneas de código, debido a que no se trata de un sistema completo, sino de una adaptación para sistemas de terceros existentes, donde aparecen paradigmas que no siguen las pautas para las cuales fueron creados modelos de estimación tipo COCOMO, y por lo tanto, encajar conceptos como puntos de función es ilógico.

6.2 Planificación temporal del proyecto

A continuación, se describe en detalle la planificación realizada para el desarrollo del proyecto. El tiempo consumido aproximado ha sido de ocho meses, con comienzo a finales de marzo de 2012 y final a principios de diciembre de 2012.

En un primer lugar, el proyecto se divide en tareas desarrolladas, coincidentes con sus fases de desarrollo generales. Estas tareas son las siguientes:

- **Propuesta del proyecto:** consiste en la presentación y descripción de la solución por parte de la empresa al desarrollador, así como su presentación y descripción como proyecto final a la universidad.

- **Análisis:** estudio de las necesidades y requisitos de la solución, así como la definición de sus características finales de manera global.

    - Definición del **problema**.

    - Definición de **requisitos**.

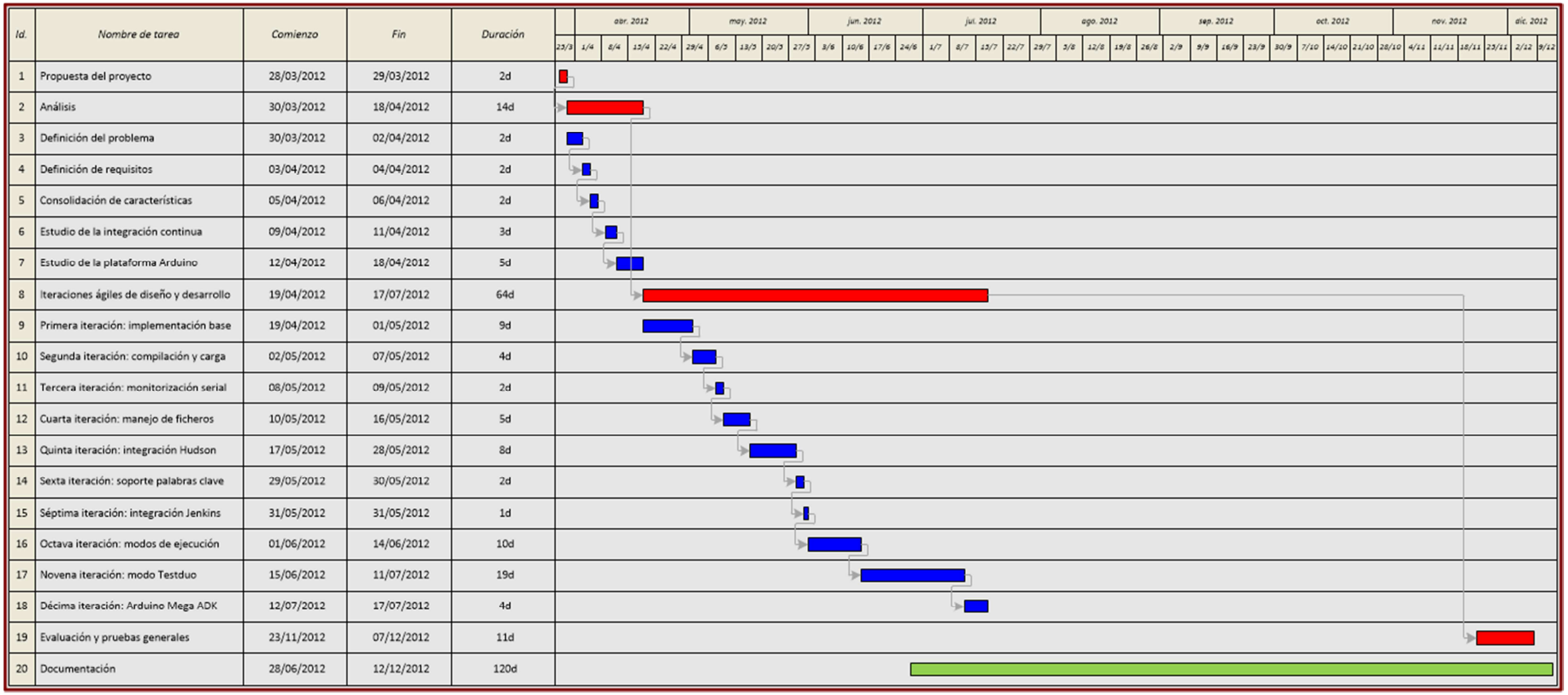
    - Consolidación de **características**.

    - **Estudio** de la **integración continua**.

    - **Estudio** de la plataforma **Arduino**.

- **Iteraciones** utilizando la metodología ágil **SCRUM**, ambas incluyen el **diseño**, la **implementación** y **pruebas** particulares de lo implementado.
    - **Primera iteración:** base de pruebas (clases y estructura de datos).
    - **Segunda iteración:** compilación y carga.
    - **Tercera iteración:** monitorización serial.
    - **Cuarta iteración:** directorios propios y mejora de manejo de ficheros.
    - **Quinta iteración:** integración con Hudson y Subversion.
    - **Sexta iteración:** soporte de palabras clave.
    - **Séptima iteración:** integración con Jenkins.
    - **Octava iteración:** soporte de diversos modos de ejecución.
    - **Novena iteración:** modo Testduo (*sketch*, librería y código Python).
    - **Décima iteración:** soporte a Arduino Mega ADK.
- **Evaluación y pruebas generales:** pruebas finales de funcionamiento, mediciones de rendimiento, comparación de modos de ejecución y estudio de impacto en Arduino.
- **Documentación:** generación de este documento.

En la página siguiente, se muestra el diagrama de Gantt correspondiente, desglosado por las tareas descritas.



Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Propuesta del proyecto	28/03/2012	29/03/2012	2d
2	Análisis	30/03/2012	18/04/2012	14d
3	Definición del problema	30/03/2012	02/04/2012	2d
4	Definición de requisitos	03/04/2012	04/04/2012	2d
5	Consolidación de características	05/04/2012	06/04/2012	2d
6	Estudio de la integración continua	09/04/2012	11/04/2012	3d
7	Estudio de la plataforma Arduino	12/04/2012	18/04/2012	5d
8	Iteraciones ágiles de diseño y desarrollo	19/04/2012	17/07/2012	64d
9	Primera iteración: implementación base	19/04/2012	01/05/2012	9d
10	Segunda iteración: compilación y carga	02/05/2012	07/05/2012	4d
11	Tercera iteración: monitorización serial	08/05/2012	09/05/2012	2d
12	Cuarta iteración: manejo de ficheros	10/05/2012	16/05/2012	5d
13	Quinta iteración: integración Hudson	17/05/2012	28/05/2012	8d
14	Sexta iteración: soporte palabras clave	29/05/2012	30/05/2012	2d
15	Séptima iteración: integración Jenkins	31/05/2012	31/05/2012	1d
16	Octava iteración: modos de ejecución	01/06/2012	14/06/2012	10d
17	Novena iteración: modo Testduo	15/06/2012	11/07/2012	19d
18	Décima iteración: Arduino Mega ADK	12/07/2012	17/07/2012	4d
19	Evaluación y pruebas generales	23/11/2012	07/12/2012	11d
20	Documentación	28/06/2012	12/12/2012	120d

Ilustración 47: Diagrama de Gantt

7. Presupuesto

En esta sección, se pretende mostrar una estimación del coste derivado del desarrollo del proyecto. Este presupuesto se encuentra dividido en varias categorías, que permiten facilitar la clasificación de los gastos, con el fin de analizar el coste final.

Todos los precios indicados excluyen el IVA, que se suma en los costes totales del proyecto, al final del presupuesto.

7.1 Costes de personal

A lo largo de los nueve meses, el desarrollo del proyecto ha sido llevado a cabo por un solo ingeniero junior. El trabajo realizado ha consumido un total de 186 días, según lo expuesto en el diagrama de Gantt del apartado de planificación, con una estimación de 6 horas de trabajo por día laborable aproximadamente. Se han tenido en cuenta como días libres de trabajo los festivos nacionales, locales, autonómicos y días de baja o vacaciones.

Por hora, el precio estimado de un ingeniero junior es de 35,50 euros. Este valor es la media de precios manejada actualmente para la posición descrita.

Teniendo en cuenta los datos aportados, el cálculo de costes de personal es el siguiente:

Categoría	Coste/Hora	Horas	Total
Ingeniero junior	35,50 €	1116	39.618,00 €
TOTAL PERSONAL			**39.618,00 €**

Tabla 171: Costes de personal

7.2 Costes de hardware

En los costes de hardware se incluyen todos los elementos hardware que han sido necesarios para el desarrollo del proyecto:

Concepto	Precio/unidad	Cantidad	Total
Workstation portátil MOUNTAIN GTM17	1.875,25 €	1	1.875,25 €
Ordenador DELL Optiplex	460,34 €	2	920,68 €
Arduino Starter Kit (with Arduino UNO)	79,90 €	3	239,70 €
Arduino UNO	20,00 €	3	60,00 €
Arduino MEGA ADK for Android	39,00 €	3	117,00 €
Telefónica I+D GSM/GPRS Shield	59,00 €	2	118,00 €
Cable macho USB tipo A - macho USB tipo B	3,60 €	9	32,40 €
Cable macho USB tipo A - macho MicroUSB tipo B	3,15 €	3	9,45 €
TOTAL HARDWARE			**3.372,48 €**

Tabla 172: Costes de hardware

7.3 Costes de software

Los precios asociados al software varían según el tipo de licencia. Las licencias gratuitas no figuran en este presupuesto.

Concepto	Precio/unidad	Cantidad	Total
Microsoft Windows 7 Enterprise**	299,95 €	2	599,90 €
Microsoft Windows 8 Pro*	29,99 €	1	29,99 €
Microsoft Office 2010 Professional	699,90 €	1	699,90 €
Microsoft Office Visio 2010 Premium	1.295,00 €	1	1.295,00 €
Adobe Photoshop CS6 Extended	1.349,00 €	1	1.349,00 €
Altova UModel 2011 Enterprise Edition	299,00 €	1	299,00 €
TOTAL SOFTWARE			**4.272,79 €**

** Licencia de empresa por volumen, el valor real se desconoce.
* Licencia académica, valor real gratuito.

Tabla 173: Costes de software

7.4 Presupuesto final

A continuación, se presenta el resultado final del presupuesto, con el coste total derivado del desarrollo del proyecto.

Primeramente, se realiza la suma de los costes obtenidos en personal, hardware y software. Posteriormente, se agrega el Impuesto sobre el Valor Añadido (IVA) vigente en España, y el coste asociado al riesgo y beneficio en porcentaje.

Concepto	Total
Costes de personal	39.618,00 €
Costes de hardware	3.372,48 €
Costes de software	4.272,79 €
Subtotal	47.263,27 €
Riesgo (10%)	4.726,33 €
Beneficio (35%)	16.542,15 €
Total sin IVA	68.531,75 €
TOTAL (21% IVA incluido)	**82.923,42 €**

Tabla 174: Presupuesto del proyecto

El presupuesto total del proyecto, antes de impuestos, asciende a la cantidad de **SESENTA Y OCHO MIL QUINIENTOS TREINTA Y UNO EUROS CON SETENTA Y CINCO CÉNTIMOS DE EURO.**

El coste total del proyecto, incluyendo el 21% de IVA vigente, es de **OCHENTA Y DOS MIL NOVECIENTOS VEINTITRÉS EUROS CON CUARENTA Y DOS CÉNTIMOS DE EURO.**

8. Conclusiones y trabajos futuros

En esta sección, se analiza la consecución de los objetivos establecidos para el proyecto y las conclusiones e ideas obtenidas a lo largo de su desarrollo. Además, también se incluyen unas conclusiones personales del desarrollador del proyecto sobre la experiencia obtenida con él. Posteriormente, se extiende un guion sobre el futuro próximo a seguir en cuanto a mejoras, innovación y líneas de investigación.

8.1 Conclusiones generales

Considerando los objetivos establecidos en el proyecto, indicados en el apartado 1.2, se pueden establecer las siguientes conclusiones como consecuencia de la consecución de los objetivos descritos:

- Se ha alcanzado la meta de implementar **una solución integrada de automatización de pruebas para software empotrado en microcontroladores**, tomando como caso Arduino UNO y Arduino Mega ADK, convirtiéndose en la **primera solución real** aportada en esta cuestión.

- Se ha logrado diseñar e implementar **una herramienta de gestión de pruebas para Arduino**, que cumple con las características establecidas en los objetivos para su cometido:

    - **Escalabilidad:** por su diseño de arquitectura de procesos paralelos, permite gestionar diferentes placas hardware Arduino en el mismo instante.

    - **Optimización:** la diferencia de rendimiento con el entorno de desarrollo oficial es aceptable, como se ha demostrado en la evaluación (apartado 5).

    - **Fiabilidad:** se incorporan opciones que eviten el bloqueo del equipo u otras tareas, además de la información detallada que se ofrece en todo momento en la salida de consola.

    - **Reusabilidad:** gracias a la política de código abierto y al estar implementado en plataformas muy conocidas y utilizadas como el lenguaje Python, es totalmente reutilizable. Además, se ha separado el código de compilación y carga en una sola clase (*MakeArduino*) para su reutilización sencilla por otro software o su uso directo por línea de comandos.

- Se ha aportado a la comunidad **una forma de testear el propio hardware,** pudiendo **monitorizar los pines** para conocer si la información que debe otorgar el software es fiable, gracias a la **librería Testduo** para Arduino, que, además, facilita la monitorización a través de objetos sencillos que no penalizan el rendimiento ni la memoria en exceso.

- Se ha establecido **un nuevo paradigma de desarrollo para el hardware empotrado**, pudiendo incorporar **métodos ágiles de desarrollo** a un escenario no habituado a ello y habilitando una forma más eficaz de **testear el comportamiento del software/firmware empotrado en el hardware directamente**.

- Se ha logrado una **integración con los sistemas de integración continua** más utilizados, a través de ejecución de comandos, soportado por la mayoría de estos sistemas. Lo cual, nos permite disponer de **la solución en conjunto con otros proyectos** de integración continua y repositorios de versiones.

Es importante recalcar que, en cuanto al desarrollo y su dificultad, Python ha sido un lenguaje muy adecuado para este tipo de aplicación, dadas las facilidades que ofrece en manejo de cadenas de caracteres. Por otra parte, la evaluación realizada ha permitido esclarecer el comportamiento de los microcontroladores de Arduino ante pruebas de rendimiento, siendo, por tanto, una documentación valiosa que escasea en internet o no es fácilmente accesible.

En cuanto a problemas encontrados, la compatibilidad de Arduino Mega ADK ha resultado un poco compleja por los parámetros que utiliza en la compilación y carga. Pero aun así, la implementación tenía una dificultad en su justa medida, sin ser demasiado complicada.

8.2 Conclusiones personales

Como desarrollador de este proyecto, concluyo que Arduino y el software empotrado en general es perfectamente compatible con metodologías de desarrollo ágil. El problema de su escasez de uso en este ámbito viene dado por la costumbre de los ingenieros, muy cercanos a la electrónica y las telecomunicaciones, donde las metodologías empleadas difieren en buena parte de las utilizadas en el software actualmente.

En cuanto al hardware, Arduino es un microcontrolador con limitaciones, pero que con una buena lógica, ya sea programada en el propio microcontrolador o en un *backend* de apoyo, es muy potente y permite realizar proyectos muy vistosos con unos conocimientos no muy amplios de programación y electrónica.

Por otro lado, los sistemas de integración continua se encuentran en plena evolución aún, y aunque cumplen con su cometido de buena manera, queda todavía camino que recorrer en cuanto a la optimización de dichos sistemas, y a la invención de técnicas de comprobación y corrección que no dependan del control exclusivo de una persona.

Como resumen final, el proyecto es muy innovador y puede marcar un punto de encuentro y de comienzo para el desarrollo de futuras soluciones integradoras que tengan en cuenta el hardware y el software empotrado en mayor medida, dada su, cada vez, mayor importancia debido a los últimos avances y productos que están saliendo al mercado (Raspberry Pi, procesadores ARM, etc.).

8.3 Líneas de continuación futuras

En esta sección, se enumeran diferentes propuestas para continuar a partir de lo implementado en este proyecto. De esta manera, se aportan ideas para expandir su desarrollo futuro:

- Construcción de un ***plugin* de Jenkins** que mejore la **integración** del programa de adaptación de Testduino con este sistema de integración continua, o incluso que reduzca o **evite la dependencia de un programa Python** externo, así como que permita y estandarice la integración de otros dispositivos hardware con software empotrado.

- **Extensión** del programa de adaptación de Testduino para **soportar más tipos de placas** Arduino tales como Arduino Leonardo, Arduino TinyPad o Arduino Due, así como **otras arquitecturas** como PIC o ARM y **otras plataformas** como OpenPICUS, Raspberry Pi, etc.

- Implementación de un procedimiento de **creación de ficheros de configuración de pruebas**, que establezcan aspectos como el orden, configuración propia, resultados posibles, fallos descubiertos, *breakpoints*, etc.

- Soporte de un **modo depuración** (*debug*) para un *sketch*, al estilo IDE típico de un lenguaje de programación, que **muestre en cada paso las variables y sus valores**.

- Mejora de la comunicación serie permitiendo el **envío de datos automático** por el puerto comunicación con la placa Arduino **cuando se requiera**, mediante previa configuración en, por ejemplo, un fichero.

- Soporte de más **comunicaciones y protocolos**: FTDI, I2C, Bluetooth, etc., para la comunicación con el hardware. Así como de **compilación y carga mediante programador**.

- Incorporación de **carga y testeo de *bootloaders*** y de **rendimiento** con diferentes *bootloaders*.

- **Soporte** de sistemas operativos tipo **UNIX**.

- **Externalización** del programa de adaptación de Testduino convirtiéndose en un **software independiente** con su propia planificación y panel de control mediante servidor web, aplicación móvil, etc.

- **Adaptaciones** expresas **a otros sistemas de integración continua** como CruiseControl.

9. Acrónimos y abreviaturas

- **2G**: Second Generation *(GSM/GPRS)*.
- **ABEL**: Advanced Boolean Expression Language.
- **ADK**: Android Development Kit.
- **ALU**: Arithmetic Logic Unit.
- **AMD**: Advanced Micro Devices.
- **ANSI**: American National Standards Institute.
- **API**: Application Programming Interface.
- **AR**: Archiver.
- **ARM**: Advanced RISC Machine.
- **ASCII**: American Standard Code for Information Interchange.
- **ASIC**: Application-Specific Integrated Circuit.
- **AVR**: Alf and Vegard RISC processor.
- **AVRDUDE**: AVR Downloader/Uploader.
- **BASIC**: Beginner's All-purpose Symbolic Instruction Code.
- **BBDD**: Bases de datos.
- **BCD**: Binary-Coded Decimal.
- **BSD**: Berkeley Software Distribution.
- **CHAR**: Character.
- **COCOMO**: Constructive Cost Model.
- **COFF**: Common Object File Format.
- **COM**: Communication Port.
- **CPP**: C Plus Plus *(fichero)*.
- **CPU**: Central Processing Unit.
- **CVS**: Concurrent Version System.
- **DLL**: Dynamic-Link Library.
- **Docutils**: Documentation Utilities.
- **DRM**: Digital Rights Management.
- **DSP**: Digital Signal Processing.
- **DWARK**: Debugging With Attributed Record Formats.
- **E/S**: Entrada y salida.
- **EC2**: *(Amazon)* Elastic Compute Cloud.
- **EEP**: EEPROM Data File.
- **EEPROM**: Electrically Erasable Programmable Read-Only Memory.
- **ELF**: Executable and Linking Format.
- **ESA**: European Space Agency.
- **EULA**: End User License Agreement.
- **EXE**: Executable.
- **FAT**: File Allocation Table.
- **FLOAT**: Floating point.
- **FPGA**: Field Programmable Gate Array.
- **FTDI**: Future Technology Devices International.

- **FTP**: File Transfer Protocol.
- **GB**: Gigabytes.
- **GCC**: GNU Compiler Collection.
- **GFLOPS**: Giga Floating-Point Operations per Second.
- **GHz**: Gigahercios.
- **GNU**: GNU Is Not UNIX.
- **GPIO**: General Purpose Input/Output.
- **GPL**: General Public License.
- **GPRS**: General Packet Radio Service.
- **GPU**: Graphical Processor Unit.
- **GSM**: Global System for Mobile communications.
- **GUI**: Graphical User Interface.
- **HD**: High Definition.
- **HDL**: Hardware Description Language.
- **HDMI**: High-Definition Multimedia Interface.
- **HEX**: Hexadecimal.
- **HP**: Hewlett-Packard.
- **HTTP**: Hyper-Text Transfer Protocol.
- **Hz**: Hercios.
- **I+D**: Investigación y Desarrollo.
- **I2C**: Inter-Integrated Circuit.
- **IBM**: International Business Machines.
- **IDE**: Integrated Development Environment.
- **INO**: Sketch Arduino *(fichero)*.
- **INT**: Integer.
- **IP**: Internet Protocol.
- **IRC**: Internet Relay Chat.
- **ICSP**: In-Circuit Serial Programming.
- **IVA**: Impuesto sobre el Valor Añadido.
- **Javadoc**: Java Documentation.
- **JSP**: JavaServer Pages.
- **Kbps**: Kilobits Per Second.
- **KB**: Kilobytes.
- **KHz**: Kilohercios.
- **LCD**: Liquid Crystal Display.
- **LED**: Light-Emitting Diode.
- **LDAP**: Lightweight Directory Access Protocol.
- **LMA**: Load Memory Address.
- **LXDE**: Lightweight X11 Desktop Environment.
- **M2M**: Machine to Machine.
- **mA**: Miliamperios.
- **MAC**: Macintosh.
- **Mb**: Megabits.

- **MCU**: Microcontroller Unit.
- **MHz**: Megahercios.
- **MIPS**: Microprocessor without Interlocked Pipeline Stages.
- **MIT**: Massachusetts Institute of Technology.
- **MMC**: MultiMediaCard.
- **MPEG**: Moving Picture Experts Group.
- **MS-DOS**: Microsoft Disk Operating System.
- **NUC**: Next Unit of Computing.
- **OBJCOPY**: Object Copy.
- **OpenGL**: Open Graphics Library.
- **OS**: Operating System.
- **OS X**: Operating System X *(anteriormente Mac OS)*.
- **PC**: Personal Computer.
- **PCB**: Printed Circuit Board.
- **PDI**: Product Development & Innovation.
- **PIC**: Peripheral Interface Controller.
- **PID**: *(USB)* Product ID.
- **PFC**: Proyecto Fin de Carrera.
- **PROM**: Programmable Read-Only Memory.
- **PWM**: Pulse Wide Mode.
- **RAM**: Random Access Memory.
- **RCA**: Radio Corporation of America.
- **RCS**: Revision Control System.
- **REST**: Representational State Transfer.
- **RISC**: Reduced Instruction Set Computing.
- **RJ45**: Registered Jack 45.
- **RMI**: Remote Method Invocation.
- **ROM**: Read-Only Memory.
- **RPC**: Remote Procedure Call.
- **RPM**: Revolutions per Minute.
- **RS-232**: Recommend Standard 232.
- **RS-485**: Recommend Standard 485.
- **RSS**: Rich Site Summary.
- **RST**: reStructuredText.
- **RTOS**: Real-Time Operating System.
- **s**: Segundos.
- **S3**: *(Amazon)* Simple Storage Service.
- **SCCS**: Source Code Control System.
- **SCM**: Source Control Management.
- **SD**: Secure Digital.
- **SDIO**: Secure Digital Input/Output.
- **SDK**: Software Development Kit.
- **SDRAM**: Synchronous Dynamic Random Access Memory

- **SEN**: Seno *(matemático)*.
- **SIM**: Subscriber Identity Module.
- **SMS**: Short Message Service.
- **SMTP**: Simple Mail Transfer Protocol.
- **SNTP**: Simple Network Time Protocol.
- **SOC**: System-On-a-Chip.
- **SPARC**: Scalable Processor Architecture.
- **SPI**: Serial Peripheral Interface.
- **SQL**: Structured Query Language.
- **SRAM**: Static Random Access Memory.
- **SSD**: Solid State Disc.
- **SSH**: Secure Shell.
- **SVK**: Subversion Kao.
- **SVN**: Subversion.
- **TCP**: Transmission Control Protocol.
- **TX/RX**: Transmission/Reception.
- **TXT**: Texto plano *(fichero)*.
- **U.S.**: United States.
- **UART**: Universal Asynchronous Receiver/Transmitter.
- **UDP**: User Datagram Protocol.
- **uFL**: Hirose U.FL connector.
- **UML**: Unified Model Language.
- **USB**: Universal Serial Bus.
- **v**: Voltios.
- **VAL**: Value.
- **VBScript**: Visual Basic Script.
- **VHDL**: *Combinación de los acrónimos* VHSIC *y* HDL.
- **VHSIC**: Very High Speed Integrated Circuit.
- **VID**: *(USB)* Vendor ID.
- **WAR**: Web Archive.
- **WebDAV**: Web Distributed Authoring and Versioning.
- **WIFI**: Wireless Fidelity.
- **X11**: X Window System v11.
- **XCOFF**: eXtended COFF.
- **XMBC**: Xbox Media Center.
- **XML**: eXtensible Markup Language.
- **µs**: Microsegundos.

10. Bibliografía

[1] Physical Internet Lab. Telefónica I+D (PDI). 2012.
Disponible: http://www.tid.es/es/labs/physicalinternet/Paginas/Purpose.aspx

[2] Introducción. Sitio web oficial de Arduino. 2012.
Disponible: http://arduino.cc/en/

[3] Telefónica I+D Shield GSM/GPRS. Sitio web oficial de BlueVia. 2012
Disponible: http://bluevia.com/en/page/tech.arduino

[4] David Astels. Test-driven development, a practical guide.
Prentice Hall, 2003. ISBN: 0-13-101649-0.

[5] James W. Grenning. Test-driven development for embedded C.
The Pragmatic Bookshelfs, 2011. ISBN: 1-934356-62-X.

[6] Enciclopedia sobre el control de versiones (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Control_de_versiones

[7] Enciclopedia sobre los sistemas de control de versiones (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Programas_para_control_de_versiones

[8] Enciclopedia sobre Concurrent Versions System (español). 2012.
Disponible: http://es.wikipedia.org/wiki/CVS

[9] Enciclopedia sobre Subversion (español). 2012
Disponible: http://es.wikipedia.org/wiki/Subversion_(software)

[10] Apache Subversion Features. Subversion website. 2012
Disponible: http://subversion.apache.org/features.html

[11] Enciclopedia sobre Git (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Git

[12] Enciclopedia sobre Microsoft Team Foundation Server (inglés). 2012.
Disponible: http://en.wikipedia.org/wiki/Team_Foundation_Server

[13] Introducción a Team Foundation Server. MSDN website. 2007.
Disponible: http://msdn.microsoft.com/es-es/library/ms181238(v=vs.90).aspx

[14] Enciclopedia sobre Plastic SCM (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Plastic_SCM

[15] Sitio web oficial de Plastic SCM. 2012.
Disponible: http://www.plasticscm.com/

[16] Enciclopedia sobre sistemas empotrados (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Sistema_embebido

[17] Enciclopedia sobre microcontroladores (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Microcontrolador

[18] Ing. Horacio. D. Vallejo. Microcontroladores AVR de Atmel. 2012.
Disponible: http://www.clubse.com.ar/DIEGO/NOTAS/2/nota18.htm

[19] Enciclopedia sobre FPGAs (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Field_Programmable_Gate_Array

[20] Tema 1. Electrónica digital. Dpto. de Tecnología Electrónica. Universidad Rey Juan Carlos. 2012.
No disponible: http://laimbio08.escet.urjc.es/assets/files/tema1.pdf.

[21] Enciclopedia sobre hardware libre (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Hardware_libre

[22] Enciclopedia sobre Arduino (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Arduino

[23] Arduino UNO Technical Specifications. Sitio web oficial de Arduino. 2012.
Disponible: http://arduino.cc/en/Main/ArduinoBoardUno

[24] Arduino Leonardo Technical Specifications. Sitio web oficial de Arduino. 2012.
Disponible: http://arduino.cc/en/Main/ArduinoBoardLeonardo

[25] Arduino Mega 2560 Technical Specifications. Sitio web oficial de Arduino. 2012.
Disponible: http://arduino.cc/en/Main/ArduinoBoardMega2560

[26] Arduino LilyPad Techinal Specifications. Sitio web oficial de Arduino. 2012.
Disponible: http://arduino.cc/en/Main/ArduinoBoardLilyPad

[27] Arduino Mega ADK Technical Specifications. Sitio web oficial de Arduino. 2012.
Disponible: http://arduino.cc/en/Main/ArduinoBoardADK

[28] Products list. Sitio web oficial de Arduino. 2012.
Disponible: http://arduino.cc/en/Main/Products

[29] Enciclopedia sobre OpenPICUS (inglés). 2012.
Disponible: http://en.wikipedia.org/wiki/Openpicus

[30] Arquitectura del microcontrolador PIC24. Sitio web oficial de Microchip. 2012.
Disponible: http://www.microchip.com/pagehandler/en-us/family/16bit/architecture/pic24f.html

[31] Flyport WiFi. OpenPICUS Wiki. Sitio web oficial de OpenPICUS. 2012.
Disponible: http://wiki.openpicus.com/index.php?title=Flyport_WiFi

[32] Flyport Ethernet. OpenPICUS Wiki. Sitio web oficial de OpenPICUS. 2012.
Disponible: http://wiki.openpicus.com/index.php?title=Flyport_Ethernet

[33] OpenPICUS Hardware. OpenPICUS Wiki. Sitio web oficial de OpenPICUS. 2012.
Disponible: http://wiki.openpicus.com/index.php?title=OpenPicus_Hardware

[34] Flyport IDE. OpenPICUS Wiki. Sitio web oficial de OpenPICUS. 2012.
Disponible: http://wiki.openpicus.com/index.php?title=OpenPicus_IDE

[35] Enciclopedia sobre Raspberry Pi (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Raspberry_Pi

[36] Downloads. Sitio web oficial de Raspberry Pi. 2012.
Disponible: http://www.raspberrypi.org/downloads

[37] Mónica Tilves. NUC, la alternativa de Intel a Raspberry Pi. SiliconWeek. 2012.
Disponible: http://www.siliconweek.es/noticias/nuc-la-alternativa-de-intel-a-raspberry-pi-22535

[38] Enciclopedia sobre integración continua (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Integraci%C3%B3n_continua

[39] Martin Fowler. Continuous Integration. Blog personal. 2006.
Disponible: http://www.martinfowler.com/articles/continuousIntegration.html

[40] Integración continua. Wiki de Dos Idea. Sitio web Dos Ideas. 2012.
Disponible: http://www.dosideas.com/wiki/Integracion_Continua

[41] Luis Rodríguez Neches. Continuous Integration is NOT the Key. Blog personal. 2012.
Disponible: http://luixrodriguezneches.wordpress.com/2012/08/27/continuous-integration-is-not-the-key/

[42] Enciclopedia sobre CruiseControl (español). 2012.
Disponible: http://es.wikipedia.org/wiki/CruiseControl

[43] CruiseControl Overview. Sitio web oficial de CruiseControl. 2012.
Disponible: http://cruisecontrol.sourceforge.net/overview.html

[44] Enciclopedia sobre Jenkins (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Jenkins

[45] Meet Jenkins. Sitio web oficial de Jenkins. 2012.
Disponible: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

[46] Enciclopedia sobre Bamboo (inglés). 2012.
Disponible: http://en.wikipedia.org/wiki/Bamboo_(software)

[47] Bamboo Feature Tour. Sitio web oficial de Atlassian. 2012.
Disponible: http://www.atlassian.com/software/bamboo/overview

[48] Enciclopedia sobre Apache Continuum (español). 2012.
Disponible: http://es.wikipedia.org/wiki/Apache_Continuum

[49] Continuum Features. Sitio web oficial de Apache. 2012.
Disponible: http://continuum.apache.org/features.html

[50] Manual de avr-gcc. Linux man page. Sitio web Die.net. 2012.
Disponible: http://linux.die.net/man/1/avr-gcc

[51] Manual de avr-ar. Linux man page. Sitio web Die.net. 2012.
Disponible: http://linux.die.net/man/1/avr-ar

[52] Manual de avr-objcopy. Linux man page. Sitio web Die.net. 2012.
Disponible: http://linux.die.net/man/1/avr-objcopy

[53] Manual de avrdude. Linux man page. Sitio web Die.net. 2012.
Disponible: http://linux.die.net/man/1/avrdude

[54] Ken Shirrif. Secrets of Arduino PWM. Blog personal. 2009.
Disponible: http://www.arcfn.com/2009/07/secrets-of-arduino-pwm.html

[55] Enciclopedia sobre arquitectura dirigida por eventos (inglés). 2012.
Disponible: http://en.wikipedia.org/wiki/Event-driven_architecture

[56] Benchmark anónimo de Mikrocontroller.net. 2012.
Disponible: http://www.mikrocontroller.net/attachment/44126/benchmark.c